PLOS ONE

RESEARCH ARTICLE

Optimal shortening of uniform covering arrays

Jose Torres-Jimenez[1]*, Nelson Rangel-Valdez[2], Himer Avila-George[3], Oscar Carrizalez-Turrubiates[4]

1 Information Technology Laboratory, CINVESTAV-Tamaulipas. Cd. Victoria, Tamaulipas 87130, México, **2** Instituto Tecnológico de Ciudad Madero, CONACYT-TecNM. Cd. Madero, Tamaulipas 89440, México, **3** Unidad de Transferencia Tecnológica Tepic, CONACYT-CICESE, Tepic, Nayarit 63173, México, **4** SVAM International de México. Cd. Victoria, Tamaulipas 87130, México

* jtj@cinvestav.mx

Citation: Torres-Jimenez J, Rangel-Valdez N, Avila-George H, Carrizalez-Turrubiates O (2017) Optimal shortening of uniform covering arrays. PLoS ONE 12(12): e0189283. https://doi.org/10.1371/journal.pone.0189283

Editor: M. Sohel Rahman, Bangladesh University of Engineering and Technology, BANGLADESH

Received: June 21, 2017

Accepted: November 23, 2017

Published: December 21, 2017

Data Availability Statement: All relevant data are within the paper and its Supporting Information files. The supporting data is additionally available at: http://www.tamps.cinvestav.mx/~oc/OSCAR.

Funding: The authors acknowledge the GENERAL COORDINATION OF INFORMATION AND COMMUNICATIONS TECHNOLOGIES (CGSTIC) at CINVESTAV for providing HPC resources on the Hybrid Cluster Supercomputer "Xiuhcoatl," that have contributed to the research results reported. The following projects have funded the research reported in this paper: 238469 - CONACYT

Abstract

Software test suites based on the concept of interaction testing are very useful for testing software components in an economical way. Test suites of this kind may be created using mathematical objects called covering arrays. A covering array, denoted by $CA(N; t, k, v)$, is an $N \times k$ array over $\mathbb{Z}_v = \{0, \ldots, v-1\}$ with the property that every $N \times t$ sub-array covers all t-tuples of $\mathbb{Z}_v^t$ at least once. Covering arrays can be used to test systems in which failures occur as a result of interactions among components or subsystems. They are often used in areas such as hardware Trojan detection, software testing, and network design. Because system testing is expensive, it is critical to reduce the amount of testing required. This paper addresses the Optimal Shortening of Covering ARrays (OSCAR) problem, an optimization problem whose objective is to construct, from an existing covering array matrix of uniform level, an array with dimensions of $(N - \delta) \times (k - \Delta)$ such that the number of missing t-tuples is minimized. Two applications of the OSCAR problem are (a) to produce smaller covering arrays from larger ones and (b) to obtain quasi-covering arrays (covering arrays in which the number of missing t-tuples is small) to be used as input to a meta-heuristic algorithm that produces covering arrays. In addition, it is proven that the OSCAR problem is NP-complete, and twelve different algorithms are proposed to solve it. An experiment was performed on 62 problem instances, and the results demonstrate the effectiveness of solving the OSCAR problem to facilitate the construction of new covering arrays.

Introduction

Functionality tests during software development demand special attention, and they are generally important for preventing malfunctions in software components. During the testing phase, it is desirable to find all errors that could arise in a software component before it is delivered to the user. If a software component has a large number of parameters, then testing it exhaustively might be expensive because of the large number of configurations that can arise from the different parameters' values; e.g., a software component with just 20 parameters of 2 different

Métodos Exactos para Construir Covering Arrays Optimos to JT-J; 2143 - Cátedras CONACYT - Fortalecimiento de las capacidades de TICs en Nayarit to HA-G; and 148784 - Fondo Mixto CONACYT y Gobierno del Estado de Nayarit, Unidad de Transferencia Tecnologica CICESE – Nayarit to HA-G.

Competing interests: The authors have declared that no competing interests exist.

values each would require 2^{20} = 1,048,576 tests. An alternative is to test the system using a small, randomly generated test suite, but in this case, there is no guarantee of the testing coverage; instead, a better choice is to use a combinatorial testing approach that provides a coverage guarantee for small test suites. This combinatorial testing approach (also called interaction testing) guarantees the coverage of all interactions of a certain size among different values of the input parameters of a software component. This approach is based on evidence presented by [1] that many errors are produced by the interactions of only a few parameter values. Specifically, the cited authors showed evidence that test suites with an interaction size of 6 are sufficient to detect all known errors in a collection of different software components.

A uniform covering array (CA), denoted by $CA(N;t, k, v)$, is a commonly used structure in interaction testing. It is an array $\mathcal{C}$ with dimensions of $N \times k$ constructed over $\mathbb{Z}_v = \{0, \ldots, v-1\}$ with the property that every $N \times t$ sub-array covers all members of $\mathbb{Z}_v^t$ at least once. The value of N is the number of rows of $\mathcal{C}$, i.e., the number of test cases; k is the number of columns or parameters; v is the number of values that each parameter can take; and t is the degree of interaction among the parameters. Because there are $\binom{k}{t}$ sets of t columns $\{c_1,\ldots,c_t\}$, the number of different t-tuples that must be covered at least once in $\mathcal{C}$ is $v^t\binom{k}{t}$. When a specific t-tuple is missing in a set of t columns $(c_1, \ldots, c_t)$, we refer to it as a missing t-wise combination (or a missing combination, for short). Below, a $CA(6; 2, 5, 2)$ in which all $2^2\binom{5}{2}$ t-wise combinations are covered at least once is shown.

$$\mathcal{C} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The covering array construction (CAC) problem is the search for the covering array number (CAN), i.e., the minimum value N for which an array $CA(N;t, k, v)$ still exists. Formally, the CAN can be defined as $CAN(t, k, v) = \min\{N|\exists\ CA(N; t, k, v)\}$.

In some theoretical studies, the following definition is adopted: $CAN(t, k, v) = O(v^t \log k)$ [2]. This definition is interesting because as the number of columns grows linearly, the number of rows grows only logarithmically. This is an advantage of such combinatorial structures because of the possibility of deriving small test suites. For instance, for a software component with 126 binary parameters, exhaustive testing would require 2^{126} tests, whereas interaction testing with strength 2 would require only 10 tests.

A complementary problem to the CAC problem is known as the test suite reduction problem (TSRP), which consists of finding, for a given array, the smallest subset of rows that covers all t-wise combinations [3]. The CAC problem is a special case of the TSRP in which the input is an array that contains all v^k distinct test cases.

For some special cases, there are algorithms that can solve the CAC problem in polynomial time:

- when $v = t = 2$ [4],
- when v is a prime power and $k \leq v + 1$ [5], and

- when $k = t + 1$ [6].

However, the CAC problem remains highly combinatorial in most cases. Moreover, some variants have been proven to be NP-complete; e.g., the work presented in [2, 7] shows the NP-completeness of the problem of extending a matrix by one row with no fewer than m missing t-wise combinations. The problem defined in the current work is also NP-complete, as proven in this paper.

Various methods have been developed to address the CAC problem. Exact methods solve it to optimality; however, they usually require exponential time to achieve their goal [8–11]. As a result of this complexity, various approximate methods have been proposed as alternatives, including recursive [4, 12, 13], algebraic [14, 15], greedy [16–20], and meta-heuristic approaches. This last category includes methods based on strategies such as genetic algorithms [21], simulated annealing [22], and tabu search [23].

These approximate algorithms can be used to build non-optimal CAs in a reasonable time; some of these algorithms depend on the quality of their inputs to produce small CAs. Most of the time, these inputs are based on matrices that are nearly CAs. The objective of the present work is to construct matrices with sufficiently few missing combinations to still be considered quasi-CAs. Such arrays are created by solving the problem known as the Optimal Shortening of Covering ARrays (OSCAR); related results were published in [24]. The OSCAR problem is relevant to the construction of CAs because it can produce smaller CAs or excellent initialization matrices for meta-heuristic algorithms for constructing CAs. The main contributions of this work are as follows. It formalizes three of the five algorithms presented in [24]. It also presents seven new approximate strategies for solving the OSCAR problem. In addition, the present work offers a complete analysis of the performance of all of the new and old algorithms, something that has not been done before. Furthermore, it proposes three new benchmarks with more than 800 OSCAR instances, which extend the range of study to matrices with strengths of $t = \{2, 3, 4, 5\}$, whereas previous works have studied only $t = 2$; these benchmarks are used as part of the experiments conducted to analyze the strategies. These experiments not only evaluate how effectively the algorithms solve the OSCAR problem but also compare the best of them against state-of-the-art strategies. These experiments provide evidence that solving the OSCAR problem using the proposed approaches enables the creation of quasi-CAs that are better than other reported initialization functions and even than the fast and versatile IPOG-F, a state-of-the-art algorithm for constructing CAs; the main result is that the arrays produced using the proposed algorithms have 90% fewer missing t-wise combinations than those generated using the other approaches considered for comparison.

This paper is organized as follows. In the problem definition section, the OSCAR problem is formally defined; its NP-completeness is proven, and some of its applications are described. In the related work section, some of the work related to initialization functions for meta-heuristics for CA construction is presented. Subsequently, the algorithms proposed in this work for solving the OSCAR problem are presented. In the experimentation section, an experiment performed to test the proposed algorithms for the construction of matrices with few missing combinations is presented. Finally, in the conclusions section, final comments regarding this work are provided.

Problem definition

Let $\mathcal{A}$ denote a $CA(N; t, k, v)$ or a quasi-$CA(N; t, k, v)$ (a quasi-CA is a matrix with a relatively small number of missing t-combinations). Then, the OSCAR problem can be defined as $\min\{\tau(\mathcal{B}_{N' \times k'}) | \mathcal{B} \text{ is a submatrix of } \mathcal{A}\}$, where $\tau(\mathcal{B})$ is a function that counts the number of missing t-wise combinations in the given array and $N' = N - \delta$ and $k' = k - \Delta$ are defined in terms

Table 1. OSCAR example, the input array $\mathcal{A} = CA(\mathbf{6}; \mathbf{2}, \mathbf{5}, \mathbf{2})$.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

https://doi.org/10.1371/journal.pone.0189283.t001

of two predefined integer values, $0 \leq \delta \leq N - v^t$ and $0 \leq \Delta \leq k - t$, which satisfy $\delta > 0 \vee \Delta > 0$. Hence, an OSCAR instance is specified by the elements $(\mathcal{A}, \delta, \Delta)$.

The search space for an OSCAR instance consists of all submatrices $\mathcal{B}$ of the given matrix $\mathcal{A}$. Accordingly, the number of feasible solutions that form such a space can be estimated to be $\binom{N}{N-\delta}\binom{k}{k-\Delta}$, where $\binom{N}{N-\delta}$ and $\binom{k}{k-\Delta}$ represent the numbers of different ways to choose subsets of rows and columns, respectively, from the original matrix $\mathcal{A}$. Throughout the remainder of this document, for a given submatrix $\mathcal{B}$, we use $\mathcal{J}_{\mathcal{R}}$ to denote the subset of rows chosen from $\mathcal{A}$ and $\mathcal{J}_{\mathcal{C}}$ to denote the subset of columns.

We present an example of a solution to the OSCAR instance specified by $\mathcal{A} = CA(6; 2, 5, 2)$ (see Table 1), $\delta = 2$, and $\Delta = 2$, for which it is feasible to construct a solution $\mathcal{B}$ (see Table 2) where $\tau(\mathcal{B}) = 0$. The solution for this instance is obtained by eliminating $\mathcal{J}_{\mathcal{R}} = \{0, 2, 4, 5\}$ and $\mathcal{J}_{\mathcal{C}} = \{2, 3, 4\}$ from $\mathcal{A}$. Because $\tau(\mathcal{B}) = 0$, the solution $\mathcal{B}$ is a $CA(4; 2, 3, 2)$.

Alternatively, the matrix $\mathcal{A}$ can be represented by another matrix $\mathcal{A}'$ with dimensions of $N \times \binom{k}{t}$. This matrix has the same number of rows as $\mathcal{A}$ and contains one column for each subset of t columns derived from $\mathcal{A}$. Each cell $a'_{i,j} \in \mathcal{A}'$ contains a value from the set $\{0, 1, \ldots, v^t - 1\}$; this value represents the t-tuple covered by row i in the subset of t columns associated with column j.

The OSCAR instance $(\mathcal{A}, \delta, \Delta) = (CA(4; 2, 3, 2), 1, 0)$ is shown in Tables 3, 4 and 5. The initial matrix $\mathcal{A}$ is shown in Table 3, the t-tuples and sets of columns are shown in Table 4, and the new matrix representation $\mathcal{A}'$ is presented in Table 5 (t-wise combinations covered).

Table 2. Solution to the OSCAR problem, $\mathcal{B} = CA(\mathbf{4}; \mathbf{2}, \mathbf{3}, \mathbf{2})$, when $\delta = 2$ and $\Delta = 2$.

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

https://doi.org/10.1371/journal.pone.0189283.t002

Table 3. An instance of the OSCAR problem, initial matrix.

	c_1	c_2	c_3
r_1	0	0	0
r_2	0	1	1
r_3	1	0	1
r_4	1	1	0
r_5	1	1	1

https://doi.org/10.1371/journal.pone.0189283.t003

Table 4. An instance of the OSCAR problem, the *t*-tuples and sets of columns.

t-tuples	set of *t* columns
0. (0,0)	$t_1 = (c_1, c_2)$
1. (0,1)	$t_2 = (c_1, c_3)$
2. (1,0)	$t_3 = (c_2, c_3)$
3. (1,1)	

https://doi.org/10.1371/journal.pone.0189283.t004

Finally, the tuple $(\mathcal{A}', \delta, \Delta)$ is used to define an instance of the OSCAR problem, and the NP-completeness of the problem can be proven based on this new representation. The remainder of this section is devoted to this proof.

The proof that the OSCAR problem is NP-complete

To demonstrate the NP-completeness of the OSCAR problem, it is necessary to show it to be equivalent to a problem that is already known to be NP-complete. For this purpose, this work presents the transformation of the maximum cover (or MAXCOVER) problem (cf. [25] for a review of this problem) into the OSCAR problem. For the proof, the previously defined notation $(\mathcal{A}', \delta, \Delta)$ for an OSCAR instance is extended to $(\mathcal{A}', \delta, \Delta, h)$, where the value h denotes an integer that supports the following question: is there a sub-array $\mathcal{B}$ of $\mathcal{A}'$ with dimensions of $(N - \delta) \times (k - \Delta)$ such that $\tau(\mathcal{B}) \leq h$? This question transforms the OSCAR problem into its decision form, which is required for this demonstration.

First, it is proven that the OSCAR problem is NP in nature. Let us begin with the case in which $\Delta = 0$, meaning that the matrix $\mathcal{B}$ is a subset of only the rows of $\mathcal{A}'$. Clearly, the size of the search space is reduced to $\binom{N}{N-\delta}$. The claim that the problem is NP in nature holds because computing the value of $\tau(\mathcal{B})$ would require time proportional to $O\left(N \times \binom{k}{t}\right)$ to examine all possible t-wise combinations, which are equal in number to the number of columns of $\mathcal{B}$. In other words, the question of whether $\tau(\mathcal{B}) \leq h$ for the OSCAR problem can be answered in polynomial time in the dimensions of $\mathcal{B}$.

Now that it has been shown that the OSCAR problem is NP in nature, let us proceed with the transformation of the NP-complete MAXCOVER problem. The objective of the MAXCOVER problem is to cover a given set $\mathcal{Q} = \{q_1, q_2, \ldots, q_l\}$, regarded as the universe. To achieve this goal, we must use a subset of $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_m\}$, where each subset $Y_i \subseteq \mathcal{Q}$, for all $1 \leq i \leq m$, is given in advance and has a size of at most C. This problem can be characterized by the tuple $(\mathcal{Q}, \mathcal{Y}, C)$ and can be transformed into an OSCAR instance $(\mathcal{A}', \delta, \Delta, l)$ as follows: a) The matrix $\mathcal{A}'$ is constructed, with $m + 1$ rows and $l + \max\{|Y_i|\} + 1$ columns. b) For $1 \leq i \leq m$ and $1 \leq j \leq l$, the value $a'_{i,j}$ of each cell is 1 if subset Y_i covers element q_j or 0 otherwise. c) For $1 \leq i \leq m$ and $j > l$, the value $a'_{i,j}$ of each cell is 0. d) For $i = m + 1$, the value $a'_{i,j}$ of each cell is 0 if $1 \leq j \leq l$ or 1 otherwise. e) The values of δ, Δ, and h are set to $m - C$, 0, and 0,

Table 5. An OSCAR instance, t-wise combinations covered.

	t_1	t_2	t_3
r_1	0	0	0
r_2	1	1	3
r_3	2	3	1
r_4	3	2	2
r_5	3	3	3

https://doi.org/10.1371/journal.pone.0189283.t005

Table 6. The MAXCOVER instance specified by $\mathcal{Q} = \{q_1, q_2, q_3, q_4, q_5\}$, $\mathcal{Y} = \{Y_1 = \{q_1, q_2, q_5\}$, $Y_2 = \{q_2, q_4, q_5\}$, $Y_3 = \{q_1, q_4, q_5\}$, $Y_4 = \{q_1, q_2, q_3\}$, $Y_5 = \{q_2, q_3, q_4\}\}$, and $C = 3$ represented as an OSCAR instance.

Row	$\mathcal{A}'$								
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
Y_1	1	1	0	0	1	0	0	0	0
Y_2	0	1	0	1	1	0	0	0	0
Y_3	1	0	0	1	1	0	0	0	0
Y_4	1	1	1	0	0	0	0	0	0
Y_5	0	1	1	1	0	0	0	0	0
Y_6	0	0	0	0	0	1	1	1	1

https://doi.org/10.1371/journal.pone.0189283.t006

respectively. The matrix $\mathcal{A}'$ can be constructed in a time of $O(lm)$, and the derived OSCAR instance is denoted by $(\mathcal{A}', \delta, 0, 0)$.

Table 6 shows an example of the transformation of the MAXCOVER problem into the OSCAR[1] problem. The following elements are used in this case:

- $\mathcal{Q} = \{q_1, q_2, q_3, q_4, q_5\}$
- $\mathcal{Y} = \{Y_1 = \{q_1, q_2, q_5\}$
- $Y_2 = \{q_2, q_4, q_5\}$, $Y_3 = \{q_1, q_4, q_5\}$, $Y_4 = \{q_1, q_2, q_3\}$, $Y_5 = \{q_2, q_3, q_4\}$
- $C = 3$

The array $\mathcal{A}'$ has dimensions of 6×9, and the δ is equal to $5 - 3 = 2$.

Finally, to complete the proof that the OSCAR problem is NP-complete, we demonstrate that the OSCAR instance $(\mathcal{A}', \delta, 0, 0)$ built from the MAXCOVER instance $(\mathcal{Q}, \mathcal{Y}, C)$ has a solution if and only if the latter has a solution. For this purpose, we start by showing that an optimal solution for $(\mathcal{A}', \delta, 0, 0)$ must include row Y_6 of $\mathcal{A}'$. This fact can be easily proven since all t-tuples must be covered in $(\mathcal{A}', \delta, 0, 0)$ and those with the value 1 in any column $j > l$ can only be covered by row Y_{m+1}.

The next step is to show that there is a solution with C subsets for the MAXCOVER instance iff there is a matrix with $C + 1$ rows that solves $(\mathcal{A}', \delta, 0, 0)$. This condition can also be easily proven. We first note that the t-tuple with value 0 is covered for any column $j \leq l$ by the row Y_{m+1}. The same tuple is also covered for any column $j > l$ by any row from $\{Y_1, \ldots, Y_m\}$. With this information, the only t-tuples that remain uncovered are those with value 1 in any column $j \leq l$. Given that during the construction of the OSCAR instance, a t-tuple with value 1 is assigned only to those rows in columns $j \leq l$ that are associated with a subset of $\mathcal{Y}$, the following claim is valid: any subset of $\mathcal{Y}$ that is formed of C elements and represents a solution for the MAXCOVER instance can also be transformed into a solution for the OSCAR instance. This claim is justified since the associated rows with the chosen C elements cover all t-tuples for any column but those with value 0 in columns $j > l$. Then, it is necessary only to add row $m + 1$ to cover the missing t-tuples. It is also true that a solution with $C + 1$ rows for the OSCAR instance is a valid solution for the equivalent MAXCOVER instance, since it is necessary only to choose those subsets of $\mathcal{Y}$ associated with the rows selected in the solution for the OSCAR instance. Finally, if one of these instances has no solution, then neither does the other; this claim holds because of the equivalence between such solutions, which has already been shown. Hence, it is demonstrated that a solution to the MAXCOVER problem implies a solution to the OSCAR problem.

Finally, any instance of the OSCAR problem for the case of $\Delta > 0$ is equivalent to $\binom{k}{k-\Delta}$ instances of the problem with $\Delta = 0$. Since it has been proven that instances of this special case are NP-complete, then the general case of the OSCAR problem is at least as complex.

Applications of the OSCAR problem

Methods of solving the OSCAR problem have the following applications: a) they can reduce the search space in the CAC problem; b) they can directly construct CAs, when there are no t-wise combinations missing in the matrices they generate; c) they can be used as initializing functions for meta-heuristics for CA construction; d) they can aid in the identification of better upper bounds for CA matrices; and e) they can be used for fine-tuning in experimental design. Each of these applications is detailed in the remainder of this section.

The OSCAR problem successfully yields a quasi-CA that has zero a small number of missing t-wise combinations. Such a situation is convenient since instead of searching for a $CA(N + \delta; t, k + \Delta, v)$ in a feasible region with a size of $O\left(\binom{v^k}{N}\right)$, corresponding to the original domain, it may be possible to construct such a CA from a relaxed region of a smaller size, $\binom{N+\delta}{\delta}\binom{k+\Delta}{\Delta}$.

The second and third applications of the OSCAR problem are related to the construction of CAs. The OSCAR problem enables the direct construction of CAs when $\tau(\mathcal{B}) = 0$, i.e., when $\mathcal{B}$ is a CA. Additionally, whenever the matrix constructed as a solution to an OSCAR instance is not a CA (i.e., the number of missing t-tuples is greater than zero), this solution can still be used indirectly for CA construction because it can serve as the initial solution for meta-heuristic algorithms. Note that the performance of a meta-heuristic for constructing CAs depends on the quality of the initial matrix. Hence, the sub-array obtained as a solution to the OSCAR problem is adequate for this purpose because it has only a few missing t-wise combinations; this is in contrast to arrays of the same size constructed using random initialization functions, which are likely to be missing a large number of the possible t-wise combinations due to their random nature. Some of the existing meta-heuristic algorithms designed for CA construction, which show dependence on the initial matrix, are reported in [21, 23, 26, 27]. It is in algorithms of this type that the OSCAR problem finds its main area of application, namely, the generation of initial matrices with few missing t-wise combinations.

The fourth application of the OSCAR problem is the identification of new upper bounds for CA matrices. Many such upper bounds have been reported in the literature. For example, the best upper bounds for some CAs can be found in the repositories of [28, 29]. In addition, some bounds on $CAN(t, k, v)$ can be found in [29]; however, the corresponding CAs have values of N that are far from optimal.

Because of the hardness of the CAC problem, the value of $CAN(t, k, v)$ for any arbitrary set of values of t, k, and v is generally unknown. However, suitable new upper bounds can be obtained from existing matrices; e.g., between $CA(174; 2, 110, 9)$ and $CA(177; 2, 117, 9)$, the upper bounds on the required numbers of columns for the cases of $N = 175$ and $N = 176$ are unknown, but it can be inferred that they should be between $111, \ldots, 116$. Because most of these upper bounds have not been shown to be optimal, the question arises as to whether other upper bounds can be found. We conclude that inputs derived by solving the OSCAR problem can be used to test potential upper bounds in order to find new bounds for $CA(N; t, k, v)$; this can be achieved through the proper selection of the values δ and Δ used to reduce the matrix size.

Some specific cases of the values of δ and Δ are as follows:

1. When $\delta > 0$ and $\Delta = 0$, i.e., only the number of rows is to be reduced, the rows that are selected to be discarded are those whose elimination results in the minimum number of missing combinations in the final array.
2. When $\delta = 0$ and $\Delta > 0$, i.e., only the columns of columns is to be reduced, the columns that are selected to be discarded are similarly those whose elimination results in the minimum number of missing t-wise combinations. However, this case makes sense only when the array $\mathcal{A}$ is not a CA.

Finally, another application of solutions to the OSCAR problem is their direct use in testing scenarios. Through the careful selection of the OSCAR problem parameters δ and Δ, it is possible to ensure that resulting sub-array has the desired numbers of rows (i.e., test cases) and columns (i.e., parameters) to produce a quasi-CA (with 90–100% coverage of the t-tuples) that provides the required level of assurance.

The proposed methodology for the construction of CAs consists of generating an initial solution for a meta-heuristic algorithm by solving an instance of the OSCAR problem; i.e., the OSCAR problem is solved to obtain the solution $\mathcal{B}$, which is then used as the initial array in a meta-heuristic algorithm.

Related work

The construction of CAs is a highly combinatorial problem that can benefit from the use of approximate algorithms to construct CAs of a desired size within a reasonable amount of time. Many researchers, instead of directing their efforts toward finding CAs with the minimum number of rows using an exact approach, have designed approximate algorithms to improve the best known *upper bound* for CAs and then reduce the gap between that bound and the CAN. These CA construction algorithms can be classified, in accordance with their characteristics, into the following types: (a) algebraic approaches, (b) exact approaches, (c) greedy approaches, (d) transformations, and (e) meta-heuristic approaches.

Algebraic methods have the characteristic that the CA construction process involves formulas or operations using mathematical objects such as vectors, finite fields, groups, and CAs with small values of t, k, and v. Some algebraic methods yield optimal constructions, including the $CA(N; 2, k, 2)$ methods of [30] and [31]; Bush's construction method for $CA(N; t, q + 1, q)$, where q is a prime or a prime power and $q \leq t$ (cf. [5]); and the zero-sum method of [6], which yields an optimal $CA(t, t + 1, v)$ for any $t \geq 2$. The main feature of these approaches is that most of them require small CAs or quasi-CAs from which to construct larger CAs.

Exact methods are exhaustive approaches for the construction of optimal CAs. Although some approaches include techniques for accelerating the search process, they generally require exponential time to complete their task, making them practical only for the construction of small optimal CAs. This category includes branch-and-bound (B&B) strategies, such as the work proposed by [10], which incorporates symmetry-breaking techniques, partial t-wise verification and fixed blocks in the bounding process, and the work of [8], which, for the generation of a non-isomorphic $CA(N; 2, k, 2)$, uses a pruning strategy based on bounds defined by the minimum ranks established in terms of the CA size.

Greedy strategies are commonly used for combinations of the parameters N, t, k, and v for which exact methods are impractical, with the basic purpose of producing a good solution in a short time. The majority of commercial and open-source tools for generating test data (including AETG [32], TCG [17], ACTS [33], IPOG-F [19], and DDA [34]) use greedy algorithms for CA construction.

Transformations generally exploit the structure of existing CAs either to make them smaller or to support other approaches, e.g., algebraic approaches, in creating smaller CAs. This task is usually performed in one of two ways: a) through the identification of redundancy or b) through the construction of submatrices. Redundancy in a CA can be identified through the permutation of rows or columns or through the changing of symbols (cf. [35], [36] and [37]). However, approaches based on the construction of submatrices provide a better basis for new CAs, and the present work can be considered to be of this type.

Finally, similar to greedy methods, meta-heuristic approaches are strategies that are not guaranteed to find a CA with the minimum number of rows. In practice, meta-heuristic methods yield very good results, but they consume more CPU time than greedy algorithms. Some meta-heuristics that have been used to solve the CAC problem include simulated annealing (SA) [22], tabu search (TS) [38], memetic algorithms (MAs) [27], and genetic algorithms (GAs) [21].

For all of the strategies described above, the main goal is the construction of CAs, i.e., matrices with zero missing t-wise combinations. However, the CA construction performance of algebraic and meta-heuristic approaches is improved when the initial matrices are quasi-CAs, i.e., when they are missing only a small number of the possible t-wise combinations. This situation raises the question of how an initial matrix should be constructed for these approaches. The answer is to use initialization functions. Hence, these initialization functions are a key element of the development of meta-heuristics for CA construction.

The main initialization functions used in state-of-the-art methods are as follows: a) random matrix initialization [21, 23, 26, 27], b) initialization with a balanced number of symbols per column [27], c) initialization through row augmentation [39], d) initialization based on submatrices [40], and e) initialization based on greedy strategies [41, 42]. The four first strategies do not consider the number of missing t-wise combinations in the construction of the initial matrix. Strategies of the last type can be used to build CAs, but they are typically larger than the required matrix size; this situation results in random discarding of rows and/or columns that is also not optimized in terms of the number of missing t-wise combinations. Hence, an alternative is to use an existing matrix of greater size and optimize the row/column reduction process until a matrix of the required size is obtained. This optimization is exactly equivalent to solving the OSCAR problem, and this work proposes a wide variety of new meta-heuristic and hybrid strategies for this purpose.

In summary, whereas CA construction approaches (e.g., exact, greedy, algebraic, meta-heuristic and transformation methods) produce matrices with no missing t-wise combinations, the strategies presented in this work solve the OSCAR problem to generate quasi-CAs. Quasi-CAs are important because they can be used as initial matrices for CA construction strategies based on algebraic and meta-heuristic methods and can thus improve the performance of these methods in the construction of new CAs.

The remainder of this section provides a more detailed introduction to some of the relevant initialization functions found in the scientific literature related to this topic. These four initialization functions will be denoted by $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, and $\mathcal{I}_4$ in this paper; Fig 1 shows an example of each one initialization function.

Each of the four initialization functions creates an array $\mathcal{C}$ with N rows and k columns, in which each cell is initialized with a symbol of the given alphabet $\{0, 1, \ldots, v-1\}$ of v symbols. The function $\mathcal{I}_1$ is presented in [21, 23, 26, 27]; this function initializes each cell c_{ij} of $\mathcal{C}_{N\times k}$ with a symbol drawn at random from the set $\{0, 1, \ldots, v-1\}$. Fig 1 (random) shows an example of the use of $\mathcal{I}_1$ to initialize a matrix $\mathcal{C}_{10\times 4}$.

The function $\mathcal{I}_2$ initializes $\mathcal{C}_{N\times k}$ with a balanced number of randomly generated symbols per column. Each column k_i, where $1 \leq i \leq k$, will contain an almost uniform distribution of

$$\text{Candidate rows}\begin{cases}\mathcal{C}\begin{cases} r_1 = \{\,1\ 2\ 0\ 2\,\} \\ r_2 = \{\,2\ 1\ 2\ 0\,\} \end{cases} \\ \begin{cases} d_1 = \{\,0\ 0\ 1\ 1\,\} \\ d_2 = \{\,2\ 0\ 0\ 1\,\} \end{cases}\end{cases} \qquad d_1\begin{cases} d(r_1,d_1)=4 \\ d(r_2,d_1)=4 \\ g(d_1,\mathcal{C})=8 \end{cases} \qquad d_2\begin{cases} d(r_1,d_2)=3 \\ d(r_2,d_2)=3 \\ g(d_2,\mathcal{C})=6 \end{cases}$$

Fig 1. Example of the Hamming distances between the two rows r_1 and r_2 that are already in the matrix C and the two candidate rows d_1 and d_2.

https://doi.org/10.1371/journal.pone.0189283.g001

the symbols {0, 1, . . ., $v-1$}. To achieve such uniformity, a symbol is generated at random for each of the N rows of column k_i, but during the random generation process, it is ensured that the first $\mathcal{R}_1 = v - \left(N - \left\lfloor \frac{N}{v} \right\rfloor v\right)$ symbols appear $\left\lfloor \frac{N}{v} \right\rfloor$ times and that the remaining $\mathcal{R}_2 = N - \left\lfloor \frac{N}{v} \right\rfloor v$ symbols appear $\left\lceil \frac{N}{v} \right\rceil$ times. For example, in a 10×4 matrix $\mathcal{C}$ with an alphabet size of $v = 3$, each of the four columns contains $\mathcal{R}_1 = 3 - \left(10 - \left\lfloor \frac{10}{3} \right\rfloor 3\right) = 2$ symbols that appear $\left\lfloor \frac{10}{3} \right\rfloor = 3$ times and $\mathcal{R}_2 = 10 - \left\lfloor \frac{10}{3} \right\rfloor 3 = 1$ symbol that appears $\left\lceil \frac{10}{3} \right\rceil = 4$ times; this situation is exemplified in Fig 1 (balanced). The use of $\mathcal{I}_2$ guarantees that each column has a balance in the cardinalities of each symbol, something that cannot be guaranteed when using $\mathcal{I}_1$. The function $\mathcal{I}_2$ is a generalization of the initialization function presented in [27] for solving the binary CAC problem using an SA approach.

The function $\mathcal{I}_3$ initializes $\mathcal{C}_{N\times k}$ one row at a time. This function generates the first row r_1 at random; i.e., each of its cells will contain a symbol randomly chosen from {0, 1, . . .,$v-1$}. Subsequently, each new row is selected from a set of two random candidate rows d_1 and d_2 and is added to $\mathcal{C}$. The chosen candidate row is the one that maximizes the Hamming distance with respect to all rows r_s that already exist in $\mathcal{C}$. The Hamming distance between two rows is equal to the number of positions at which the corresponding symbols are different; correspondingly, the Hamming distance between a candidate row d_j and all rows already in $\mathcal{C}$ is equal to the number of positions l in each row r_s that differ from the corresponding positions in d_j, summed over all existing rows r_s. Formally, this latter definition can be expressed as $g(d_j, \mathcal{C}) = \sum_{s=0}^{i-1} \sum_{l=0}^{k-1} h(r_{s,l}, d_{j,l})$, where i is the number of rows already added to $\mathcal{C}$ and $h(r_{s,l}, d_{j,l}) = 1$ if $r_{s,l} \neq d_{j,l}$ or 0 otherwise. This process is repeated until all N rows have been created. This initialization function has been used previously in [39].

An example of the selection of a row as defined in $\mathcal{I}_3$ is shown in Fig 2; the matrix $\mathcal{C}$ already contains 2 rows, and the third row will be the candidate d_1 because it maximizes the value of $g(d_j, \mathcal{C})$. Fig 1 (Hamming) shows the full initial matrix.

Finally, the function $\mathcal{I}_4$ initializes $\mathcal{C}_{N\times k}$ based on groups of t columns. This function is based on the sub-array $\mathcal{C}' = CA(v^t; t, t, v)$, which is constructed using the v^t combinations of symbols derived from an alphabet of size v and a strength value of t; e.g., $\mathcal{C}' = CA(3^2; 2, 2, 3)$ will be formed of the elements in the set {00, 01, 02, 10, 11, 12, 20, 21, 22}, where each element represents a row in $\mathcal{C}'$. The function $\mathcal{I}_4$ is performed in two steps. In the first step, $\mathcal{C}'$ is used to define the symbols in the first t columns of the matrix $\mathcal{C}$. During this process, juxtaposition of $\mathcal{C}'$ is applied to complete the N rows of $\mathcal{C}$; specifically, $\mathcal{C}'$ is juxtaposed $\left\lfloor \frac{N}{v^t} \right\rfloor v^t$ times, and the remaining $N - \left\lfloor \frac{N}{v^t} \right\rfloor v^t$ rows of $\mathcal{C}$ are filled with the first rows of $\mathcal{C}'$. In the second step, the first t columns of $\mathcal{C}$ are copied into the next subset of t columns whose symbols have not yet been defined, and the values are changed in some pairs of rows; these changes are executed by randomly choosing $\left\lceil \frac{N}{2} \right\rceil$ pairs of rows and, for each pair, exchanging the values of those columns in each row. This step is repeated until all k columns of $\mathcal{C}$ have been defined. If the number of columns in the last subset (t') is smaller than t, then only the first t' columns of $\mathcal{C}'$ are used.

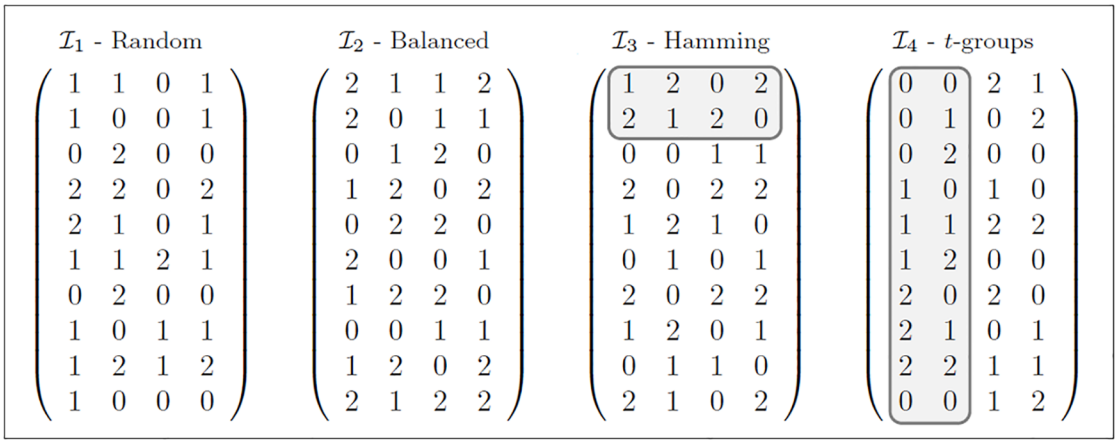

Fig 2. Initialization functions. (a) $\mathcal{I}_1$ results in 20 missing combinations. (b) $\mathcal{I}_2$ results in 18 missing combinations. (c) $\mathcal{I}_3$ results in 15 missing combinations. (d) $\mathcal{I}_4$ results in 7 missing combinations.

https://doi.org/10.1371/journal.pone.0189283.g002

This function is a generalization of the last initialization function presented in [40]. An example of this initialization method is shown in Fig 1 (t-groups).

Algorithms for solving the OSCAR problem

This paper has formally defined the OSCAR problem and has proven that it is NP-complete. Now, various strategies are proposed for solving this problem. This section is devoted to this purpose; throughout the remainder of the section, each proposed approach is described in detail.

Given that a solution to a specific instance of the OSCAR problem is defined by two sets, $\mathcal{J}_{\mathcal{R}}$ and $\mathcal{J}_{\mathcal{C}}$, and considering that each of these two sets can be selected using one of three different approaches (exact ($\mathcal{E}$), greedy ($\mathcal{G}$), or meta-heuristic ($\mathcal{M}$)), it is possible to define 9 basic algorithms, as shown in Table 7. The superindices for the $\mathcal{EE}$ and $\mathcal{GG}$ options indicate the number of variants that have been defined. For the $\mathcal{EE}$ approach, the two corresponding algorithms are denoted by $\mathcal{EE}_{\mathcal{CR}}$ (first the number of columns is reduced, then the number of rows) and $\mathcal{EE}_{\mathcal{RC}}$ (first the number of rows is reduced, then the number of columns). Three variants have been defined for the $\mathcal{GG}$ approach; these variants are denoted by $\mathcal{GG}_{\mathcal{CR}}$ (first the number of columns is reduced, then the number of rows), $\mathcal{GG}_{\mathcal{RC}}$ (first the number of rows is reduced, then the number of columns), and $\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$ (the numbers of columns and rows are reduced in an alternating fashion). Thus, we ultimately present a total of 12 possible algorithms for solving the OSCAR problem.

We first describe the reduction of the numbers of rows and columns of the initial matrix $\mathcal{A}$ using the greedy approach. Afterward, the three greedy algorithms $\mathcal{GG}_{\mathcal{RC}}$, $\mathcal{GG}_{\mathcal{CR}}$ and $\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$ are

Table 7. Different algorithms for solving the OSCAR problem. The algorithms are grouped by the exact ($\mathcal{E}$), greedy ($\mathcal{G}$), and meta-heuristic ($\mathcal{M}$) approaches.

R	C		
	E	G	M
E	$\mathcal{EE}^2$	$\mathcal{EG}$	$\mathcal{EM}$
G	$\mathcal{GE}$	$\mathcal{GG}^3$	$\mathcal{GM}$
M	$\mathcal{ME}$	$\mathcal{MG}$	$\mathcal{MM}$

https://doi.org/10.1371/journal.pone.0189283.t007

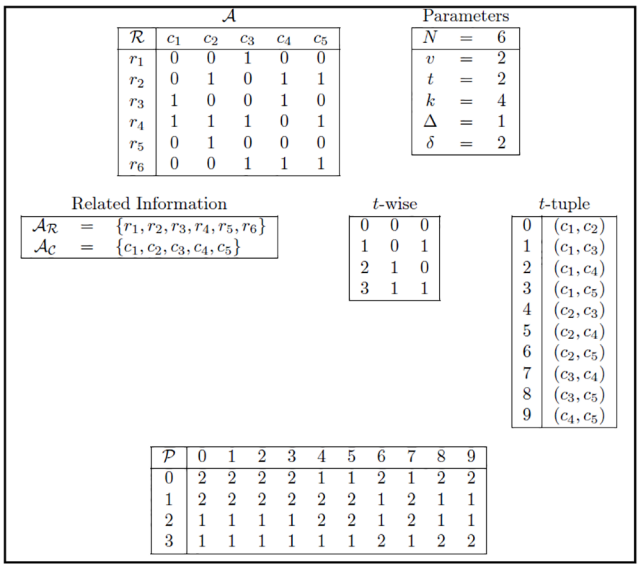

Fig 3. OSCAR instance. Problem instance specified by the matrix $\mathcal{A} = CA(6; 2, 5, 2)$ and the values $\Delta = 1$ and $\delta = 2$. Related information, t-wise combinations, t-tuples, and P matrix.

https://doi.org/10.1371/journal.pone.0189283.g003

defined. Next, we introduce the exact algorithms $\mathcal{EE}_{\mathcal{RC}}$ and $\mathcal{EE}_{\mathcal{CR}}$ for solving the OSCAR problem by exploring the entire search space, which has a size of $\binom{N}{N-\delta}\binom{k}{k-\Delta}$; these algorithms are based on a B&B approach [43]. Next, the meta-heuristic algorithm for solving the OSCAR problem is presented; this algorithm is based on the SA approach. Finally, the six hybrid algorithms $\mathcal{GE}$, $\mathcal{EG}$, $\mathcal{GM}$, $\mathcal{MG}$, $\mathcal{ME}$ and $\mathcal{EM}$ for solving the OSCAR problem are defined.

Greedy algorithms $\mathcal{GG}_{\mathcal{CR}}$, $\mathcal{GG}_{\mathcal{RC}}$, and $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$ for solving the OSCAR problem

The proposed greedy algorithms are based on two functions ($\mathcal{F}_\mathcal{R}$ and $\mathcal{F}_\mathcal{C}$) that reduce the number of rows or columns one element (row or column) at a time, starting from the array $\mathcal{A}$, while considering the number of missing t-wise combinations after the reduction process. Examples of all of the greedy strategies proposed in this section are presented based on the OSCAR instance shown in Fig 3. It presents the problem instance specified by the matrix $\mathcal{A} = CA(6; 2, 5, 2)$ and the values $\Delta = 1$ and $\delta = 2$. It shows each combination of columns, or each t-tuple, that is derived from $\mathcal{A}$ and all of the possible t-wise combinations of symbols that could be found in each of them; it also shows the sets A_R and A_C of rows and columns, respectively. Besides, it shows the auxiliary structure P, which is used to store the number of times that each t-wise combination is covered in each t-tuple; this structure P is a matrix of $v^t = 2^2$ rows and $\binom{k}{t} = \binom{5}{2} = 10$ columns, in which each cell $p_{i,j}$ contains the number of times that the i^{th} t-wise combination of symbols appears in A in the subset of columns defined by the j^{th} t-tuple.

Greedy approach for reducing the number of rows. The greedy function that reduces the number of rows is denoted by $\mathcal{F}_\mathcal{R}$, and it is defined below. Let $\mathcal{A}_\mathcal{R} = \{r_1, r_2, ..., r_N\}$ be the set of rows of $\mathcal{A}$, and let $\mathcal{O} = \{o_1, o_2, \ldots, o_N\}$ be a vector in which each element is associated with a row r_i and has a value equal to the number of t-wise combinations that are exclusively covered by the associated row. The function $\mathcal{F}_\mathcal{R}$ selects the row $\{r_i | i = \min_j\{o_j\}\}$ to be discarded; ties are broken randomly.

The function $\mathcal{F}_\mathcal{R}$ uses the vector $\mathcal{O}$ that describes the initial array $\mathcal{A}$ to choose a row r_i to be discarded such that the value o_i is minimized. Discarding that row from $\mathcal{A}$ results in an array

Table 8. Greedy approach for reducing the number of rows: Examples of the `getN()` and $\mathcal{F}_{\mathcal{R}}(\mathcal{O})$ functions. Results of $\texttt{getN}(\mathcal{A})$.

	o_1	o_2	o_3	o_4	o_5	o_6
$\mathcal{O}$	2	2	6	6	2	2

https://doi.org/10.1371/journal.pone.0189283.t008

$\mathcal{A}'$ such that $\tau(\mathcal{A}') = \tau(\mathcal{A}) + o_i$, since once row r_i is discarded, the t-wise combinations that were covered exclusively by row r_i are no longer covered in $\mathcal{A}'$. Therefore, the resulting array $\mathcal{A}'$ without row r_i will be missing the minimum possible number of t-wise combinations because o_i has the minimum value among the elements of $\mathcal{O}$. When $o_i = 0$, row r_i is clearly superfluous, since it does not cover any t-wise combinations exclusively.

Every time that a row r_i is discarded in the reduction process, the number of rows that cover each of the t-wise combinations covered by the discarded row r_i must be decreased by one. Whenever a t-wise combination is then covered by only a single remaining row j, the value of o_j must be increased by one. We update the vector $\mathcal{O}$ in this way.

The time required to initially populate $\mathcal{O}$ for the function $\mathcal{F}_{\mathcal{R}}$ is $O\left(\left(N + v^t + v^t\binom{N}{2}\right)\binom{k}{t}\right)$ (in all algorithms with a greedy component, this process is called `getN()`), since it is necessary to explore all rows per set of t columns, to determine the number of times that each t-tuple is covered, and to confirm the t-wise combinations that are covered by only one row. The time required to discard a row and update $\mathcal{O}$ is $O\left(N + (N-1)\binom{k}{t}\right)$, since it is necessary to first explore the vector $\mathcal{O}$ and then, for each set of t columns, verify the number of times that each t-tuple is covered in at most $N - 1$ rows.

Tables 8, 9 and 10 illustrate the application of $\texttt{getN}(\mathcal{A})$ and $\mathcal{F}_{\mathcal{R}}(\mathcal{O})$ to the matrix $\mathcal{A}$ defined in Fig 3. The vector $\mathcal{O}$ shown in Table 8 is the result of the call to $\texttt{getN}(\mathcal{A})$. Each element of this vector has a value equal to the number of unique t-wise combinations covered by the corresponding row; e.g., the value $o_1 = 2$ implies that row r_1 contains two t-wise combinations that are exclusively covered by this row (these are the symbol combinations 00 and 10 corresponding to the t-tuples (c_2, c_4) and (c_3, c_5), respectively). Now, a call to $\mathcal{F}_{\mathcal{R}}(\mathcal{O})$ will result in an arbitrary selection from among the rows $\{r_1, r_2, r_5, r_6\}$; let us assume that r_2 is chosen. The elimination of this row will produce the new vector $\mathcal{O}$ shown in Table 10. To illustrate the update operation of $\mathcal{F}_{\mathcal{R}}$, Table 9 shows how the auxiliary structure P is modified in accordance with the t-wise combinations that are eliminated with the deletion of row r_2; note that there are 8 new t-wise combinations that are now uniquely covered in the remaining rows. In the new

Table 9. Greedy approach for reducing the number of rows: Examples of the `getN()` and $\mathcal{F}_{\mathcal{R}}(\mathcal{O})$ functions. P matrix.

t-tuple			(c_1, c_2)	(c_1, c_3)	(c_1, c_4)	(c_1, c_5)	(c_2, c_3)	(c_2, c_4)	(c_2, c_5)	(c_3, c_4)	(c_3, c_5)	(c_4, c_5)
0	0	0	2	2 → 1	2	2	1	1	2	1	2	2
1	0	1	2 → 1	2	2 → 1	2 → 1	2	2	1	2 → 1	1 → 0	1
2	1	0	1	1	1	1	2 → 1	2	1	2	1	1
3	1	1	1	1	1	1	1	1 → 0	2 → 1	1	2	2 → 1

https://doi.org/10.1371/journal.pone.0189283.t009

Table 10. Greedy approach for reducing the number of rows: Examples of the `getN()` and $\mathcal{F}_{\mathcal{R}}(\mathcal{O})$ functions. Results of $\mathcal{F}_{\mathcal{R}}(\mathcal{O})$.

	o_1	o_3	o_4	o_5	o_6
$\mathcal{O}$	2	7	7	5	5

https://doi.org/10.1371/journal.pone.0189283.t010

vector $\mathcal{O}$, the value of the element corresponding to each of these rows is incremented by the number of t-wise combinations in that row for which the corresponding value in P has been changed to 1 after the elimination of row r_2. For example, the t-wise combinations 01, 00 and 10 associated with t-tuples (c_1, c_2), (c_1, c_3), and (c_2, c_3) are newly exclusively covered by row r_5 after the removal of row r_2; consequently, o_5 is increased from 2 to 5 in the new vector $\mathcal{O}$.

Greedy approach for reducing the number of columns. The function that reduces the number of columns using the greedy approach is denoted by $\mathcal{F}_c$ and is defined below. Let $\mathcal{A}_c$ be the set of columns of $\mathcal{A}$; let $\mathcal{K}$, with dimensions of $k \times k$, be an array in which each element $k_{i,j}$ stores the number of times that columns i and j together are involved in a missing t-wise combination; and let $\mathcal{U} = \{u_1, u_2, \ldots, u_k\}$ be a vector in which $u_i = \sum_{j=1}^{k} k_{i,j}$. The function $\mathcal{F}_c$ selects the column $\{c_i | i = \max_j\{u_j\}\}$ to be discarded; ties are broken randomly. Whenever a column i is discarded, the vector $\mathcal{U}$ is updated by subtracting the value $k_{i,j}$ from u_j for all $j \neq i$. Each element in $\mathcal{U}$ is associated with a column, and its value is equal to the number of times that column is involved in a missing t-wise combination.

In summary, the function $\mathcal{F}_c$ chooses a column i associated with the maximum value u_i in the vector $\mathcal{U}$. When discarding column i, we obtain an array $\mathcal{A}'$ such that $\tau(\mathcal{A}') = \sum_{j=0}^{k-1} u_j$, since once column i has been discarded, the associated missing combinations involving column i are deleted. Therefore, the resulting array $\mathcal{A}'$ will have the minimum number of missing t-wise combinations, since u_i has the greatest value among the elements of $\mathcal{U}$.

When we discard a column, the values of the elements of $\mathcal{U}$ must be updated. To do so, a value of −1 is assigned to u_i, and the value of each element u_j such that $j \neq i$ is updated based on its interaction with the recently discarded column; i.e., $u_j = u_j - k_{i,j}$. This process is intuitively illustrated as follows. Suppose that we have a set of n criminals who are accused of having committed m crimes together, and suppose that the authorities have found that a certain criminal s is the only one who committed l of these crimes, where $l \leq m$; then, the number of crimes of which each of the remaining criminals is accused must be decreased in accordance with his initially suspected degree of participation in committing crimes with criminal s.

The time required to initially populate $\mathcal{U}$ and $\mathcal{K}$ for the function $\mathcal{F}_c$ is $O\left(\left(N + \binom{t}{2} + t\right)\binom{k}{t}\right)$ (in all algorithms with a greedy component, this process is called `getK()`), since for each set of t columns, all N rows must be explored, the vector $\mathcal{G}$ must then be updated based on the missing t-wise combinations in these columns, and $\mathcal{K}$ must be updated for all possible pairs in this set of t columns. The time required to discard a column and update $\mathcal{U}$ and $\mathcal{K}$ accordingly is $O(2k)$, since the vector $\mathcal{U}$ must be explored to obtain the column i with the greatest value, and column i of $\mathcal{K}$ must then be explored to update $\mathcal{U}$.

Tables 11 and 12 illustrate the application of $\texttt{getk}(\mathcal{A})$ and $\mathcal{F}_c(\mathcal{K}, k, \mathcal{U})$ to the matrix $\mathcal{A}$ defined in Fig 3. The matrix $\mathcal{K}$ and the vector $\mathcal{U}$ shown in Table 11 are the results of the call to $\texttt{getk}(\mathcal{A})$; given that the initial matrix $\mathcal{A}$ is a CA, all values in $\mathcal{K}$ and $\mathcal{U}$ are zero because there

Table 11. Greedy approach for reducing the number of columns: Examples of the $\texttt{getk}(\mathcal{A})$ and $\mathcal{F}_c(\mathcal{K}, \boldsymbol{k}, \mathcal{U})$ functions.

$\mathcal{R}$	$\mathcal{K}$					$\mathcal{U}$	
	c_1	c_2	c_3	c_4	c_5	u_1	0
c_1	-	0	0	0	0	u_2	0
c_2	0	-	0	0	0	u_3	0
c_3	0	0	-	0	0	u_4	0
c_4	0	0	0	-	0	u_5	0
c_5	0	0	0	0	-		

https://doi.org/10.1371/journal.pone.0189283.t011

Table 12. Greedy approach for reducing the number of columns: New vector $\mathcal{U}$.

	u_1	u_3	u_4	u_5
U_{update}	0	0	0	0

https://doi.org/10.1371/journal.pone.0189283.t012

are no missing t-wise combinations. Now, a call to $\mathcal{F}_{\mathcal{C}}(\mathcal{K}, k, \mathcal{U})$ will result in the arbitrary selection of a column from among $\{c_1, c_2, c_3, c_4, c_5\}$; let us assume that c_2 is chosen. The elimination of this column will produce the new vector $\mathcal{U}$ shown in Table 12, which also has zero missing t-wise combinations because the values c_j, for $j \neq 2$, are all zero.

Now that the greedy functions $\mathcal{F}_{\mathcal{R}}$ and $\mathcal{F}_{\mathcal{C}}$ have been defined, the three greedy algorithms are introduced below.

Greedy algorithm $\mathcal{GG}_{\mathcal{CR}}$. $\mathcal{GG}_{\mathcal{CR}}$ is a greedy algorithm that first reduces $\mathcal{A}$ to a matrix with $k - \Delta$ columns using the function $\mathcal{F}_{\mathcal{C}}$. The newly formed array is denoted by $\mathcal{A}'$ and has N rows and $k - \Delta$ columns. Then, $\mathcal{A}'$ is further reduced to a matrix with $N - \delta$ rows using the function $\mathcal{F}_{\mathcal{R}}$, yielding the solution $\mathcal{B}$. Algorithm 1 describes the $\mathcal{GG}_{\mathcal{CR}}$ approach.

Algorithm 1

```
1: function GG_CR(A, δ, Δ)
2:     J_R ← ∅
3:     J_C ← ∅
4:     GETK(A)
5:     for i < Δ do
6:         c ← F_C(K, k, U)
7:         J_C ← J_C + c
8:     end for
9:     A' ← A_C − J_C
10:    GETN(A')
11:    for i < δ do
12:        r ← F_R(O)
13:        J_R ← J_R + r
14:    end for
15:    A' ← A_R − J_R
16:    B' ← A'
17:    return (c, B)
18: end function
```

The time required to execute $\mathcal{GG}_{\mathcal{CR}}$ can be calculated from the times required for populating and updating the necessary structures, as follows:
$O\left(\left(N + \binom{t}{2} + t\right)\binom{k}{t} + \Delta(2k) + \left(N + v^t + v^t\binom{N}{2}\right)\binom{k-\Delta}{t} + \delta\left(N + (N-1)\binom{k-\Delta}{t}\right)\right)$.

During the execution of this algorithm, Δ columns are first discarded from $\mathcal{A}$ following the defined reduction process, resulting in an array $\mathcal{A}'$ with $k - \Delta$ columns; then, the necessary structures for eliminating rows from $\mathcal{A}'$ are populated, and finally, δ rows are discarded following the defined reduction process, yielding the solution $\mathcal{B}$.

Fig 4 shows an example of the application of $\mathcal{GG}_{\mathcal{CR}}$ to the problem instance presented in Fig 3. This table illustrates how the initial matrix $\mathcal{A}$ evolves into the final matrix $\mathcal{B}$. First, Fig 4(a) shows the changes made to $\mathcal{A}$ due to the elimination of columns (see the loop in lines 5 to 8); for each iteration i of this loop, the table presents the initial vector $\mathcal{U}$, the set J_C of columns chosen so far, the vector $\mathcal{U}_{new}$ obtained by updating $\mathcal{U}$ after the elimination of the column c selected in that iteration, and the resulting matrix $\mathcal{A}'$ after that iteration. This part of the algorithm is performed only once because $\Delta = 1$. Subsequently, Fig 4(b) presents the changes made to the last matrix $\mathcal{A}'$ obtained in the previous process due to the elimination of rows (see the loop in lines 11 to 14); for each iteration i of this loop, the table presents the vector

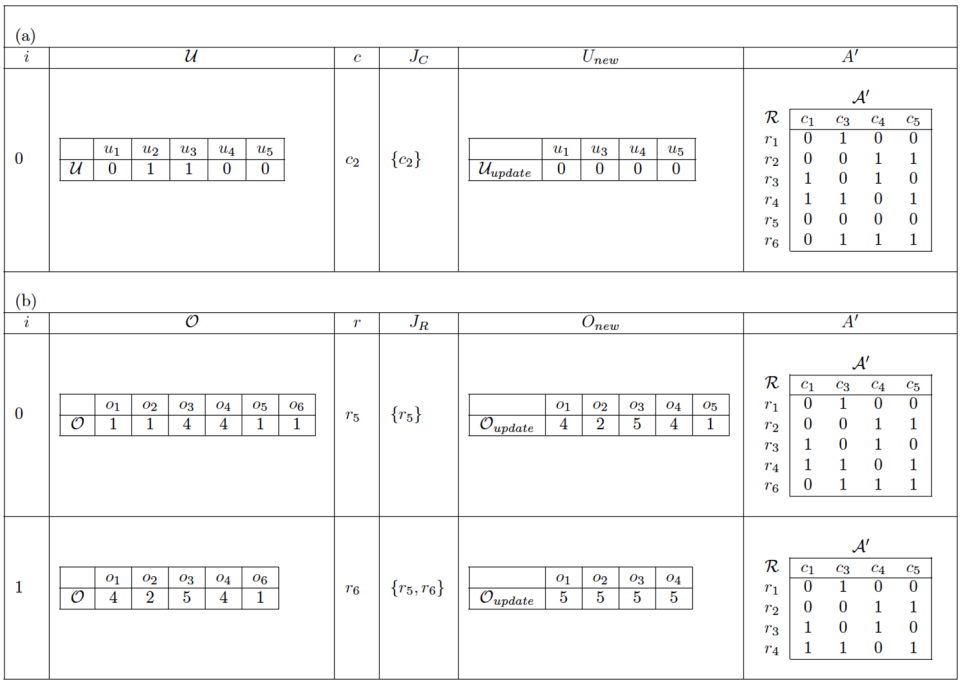

Fig 4. Example of $\mathcal{GG}_{\mathcal{CR}}$. (a) Discarding columns. (b) Discarding rows.

https://doi.org/10.1371/journal.pone.0189283.g004

$\mathcal{O}$ derived from the previous matrix $\mathcal{A}'$, the set J_R of rows chosen so far, the updated vector $\mathcal{O}_{new}$ obtained after the elimination of the row r selected in that iteration, and the resulting matrix $\mathcal{A}'$ after that iteration. This second loop is repeated twice because $\delta = 2$. The last matrix $\mathcal{A}'$ obtained in the second part of $\mathcal{GG}_{\mathcal{CR}}$ is returned as the final matrix $\mathcal{B}$.

Greedy algorithm $\mathcal{GG}_{\mathcal{RC}}$. The greedy algorithm $\mathcal{GG}_{\mathcal{RC}}$ first removes δ rows from $\mathcal{A}$ using the function $\mathcal{F}_{\mathcal{R}}$ to obtain an array $\mathcal{A}'$ with $N - \delta$ rows and k columns. Then, $\mathcal{A}'$ is reduced to a matrix with $k - \Delta$ columns using the function $\mathcal{F}_{\mathcal{C}}$ to obtain the final solution $\mathcal{B}$. Algorithm 2 describes the $\mathcal{GG}_{\mathcal{RC}}$ approach.

Algorithm 2

```
function GG_RC(A, δ, Δ)
    J_R = ∅
    J_C = ∅
    GETN(A)
    for i < δ do
        r ← F_R(O)
        J_R = J_R + r
    end for
    A' ← A_R − J_R
    GETK(A')
    for i < Δ do
        c ← F_C(K, k, U)
        J_C ← J_C + c
    end for
    A' ← A_C − J_C
    B ← A'
    return (c, B)
end function
```

The time required to execute $\mathcal{GG}_{\mathcal{RC}}$ can be calculated from the times required for populating and updating the necessary structures. The result is $O\big(\big(N + v^t + v^t\binom{N}{2}\big)\binom{k}{t} + \delta\big(N + N - 1\binom{k}{t}\big) + \big(N + \binom{t}{2}\big)\binom{k}{t}\big) + \Delta(2k)\big)$, since $N - \delta$ rows are first discarded from the input array $\mathcal{A}$, generating an array $\mathcal{A}'$ with $N - \delta$ rows and k columns, and this array is then reduced to one with $k - \Delta$ columns to obtain the solution $\mathcal{B}$.

Fig 5 shows an example of the application of $\mathcal{GG}_{\mathcal{RC}}$ to the problem instance presented in Fig 3. This table illustrates how the initial matrix $\mathcal{A}$ evolves into the final matrix $\mathcal{B}$. First, Fig 5(a) presents the changes made to $\mathcal{A}$ due to the elimination of rows (see the loop in lines 5 to 8); for each iteration i of this loop, the table presents the initial vector $\mathcal{O}$, the set J_R of rows chosen so far, the vector $\mathcal{O}_{new}$ obtained by updating $\mathcal{O}$ after the elimination of the row r selected in that iteration, and the resulting matrix $\mathcal{A}'$ after that iteration. This part of the algorithm is repeated twice because $\delta = 2$. Subsequently, Fig 5(b) presents the changes made to the last matrix $\mathcal{A}'$ obtained in the previous process due to the elimination of columns (see the loop in lines 11 to 14); for each iteration i of this loop, the table presents the vector $\mathcal{U}$ derived from the last matrix $\mathcal{A}'$, the set J_C of columns chosen so far, the updated vector $\mathcal{U}_{new}$ after the elimination of the column c selected in that iteration, and the resulting matrix $\mathcal{A}'$ after that iteration. This second loop is executed only once because $\Delta = 1$. The last matrix $\mathcal{A}'$ obtained in the second part of $\mathcal{GG}_{\mathcal{RC}}$ is returned as the final matrix $\mathcal{B}$.

Greedy algorithm $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$. The greedy algorithm $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$ distributes the elimination of Δ columns and δ rows in a round-robin fashion. This algorithm alternately discards first a single row and then some number of columns until the number of rows has been reduced to $N - \delta$. This algorithm uses a vector $\mathcal{D}$ with δ elements, where each element d_i, corresponding to the i^{th} discarded row, indicates the number of columns that should be discarded immediately after discarding that row. When $\Delta > \delta$, the first $\delta - 1$ elements of $\mathcal{D}$ are each filled with a value of $\lfloor\frac{\Delta}{\delta}\rfloor$, and the last one is filled with a value of $\lceil\frac{\Delta}{\delta}\rceil$. When $\delta \geq \Delta$, the first Δ elements of $\mathcal{D}$ are each

(a)

i	$\mathcal{O}$	r	J_R	O_{new}	A'
0	$\mathcal{O}$: o_1=2, o_2=2, o_3=6, o_4=6, o_5=2, o_6=2	r_2	$\{r_2\}$	$\mathcal{O}_{update}$: o_1=2, o_3=7, o_4=7, o_5=5, o_6=5	$\mathcal{A}'$: r_1=0 0 1 0 0; r_3=1 0 0 1 0; r_4=1 1 1 0 1; r_5=0 1 0 0 0; r_6=0 0 1 1 1
1	$\mathcal{O}$: o_1=2, o_3=7, o_4=7, o_5=5, o_6=5	r_1	$\{r_1, r_2\}$	$\mathcal{O}_{update}$: o_3=8, o_4=8, o_5=8, o_6=8	$\mathcal{A}'$: r_3=1 0 0 1 0; r_4=1 1 1 0 1; r_5=0 1 0 0 0; r_6=0 0 1 1 1

(b)

i	$\mathcal{U}$	c	J_C	U_{new}	A'
0	$\mathcal{U}$: u_1=0, u_2=2, u_3=2, u_4=2, u_5=2	c_3	$\{c_3\}$	$\mathcal{U}_{update}$: u_1=0, u_2=2, u_4=2, u_5=0	$\mathcal{A}'$: r_3=1 0 1 0; r_4=1 1 0 1; r_5=0 1 0 0; r_6=0 0 1 1

Fig 5. Example of $\mathcal{GG}_{\mathcal{RC}}$. (a) Discarding rows. (b) Discarding columns.

https://doi.org/10.1371/journal.pone.0189283.g005

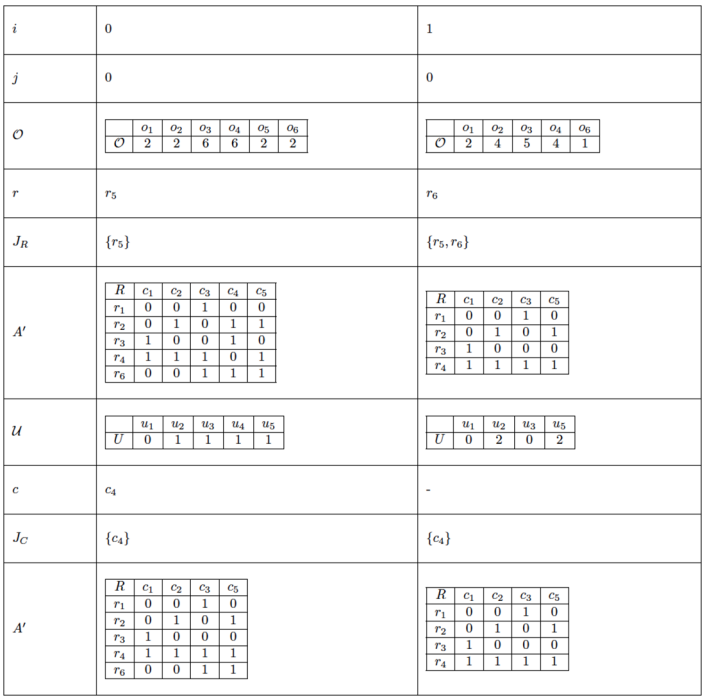

Fig 6. Example of $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$. Column 1 shows the main structures used throughout the algorithm, and each of the remaining columns represents a different iteration of the main algorithm.

https://doi.org/10.1371/journal.pone.0189283.g006

filled with a value of one. Once the Δ columns have been distributed among the δ rows, one row is discarded, and then, the number of columns is reduced to $k - d_i$. Hence, with the exploration of each element i of the vector $\mathcal{D}$, the numbers of rows and columns of the array $\mathcal{A}'$ will be decreased to $N_i = N - i$ and $k_i = k - \sum_{j=1}^{i-1} d_j$, respectively. Algorithm 3 describes the $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$ approach.

The time required to execute $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$ can be obtained by considering how the numbers of columns and rows of $\mathcal{A}'$ will be reduced while exploring the vector $\mathcal{D}$; the result is $O(\sum_{i=1}^{\delta}(N_i + v^t + (v^t\binom{k_i}{t}))\binom{k_i}{t} + (N_i + (N_i - 1\binom{k_i}{t}) + (N_i + \binom{t}{2} + t)\binom{k_i}{t} + d_i(2(k_i))$. As each element i of $\mathcal{D}$ is explored, first, the necessary structures are populated to discard rows from the array $\mathcal{A}'$, which has $N - i + 1$ rows and $k - \sum_{j=1}^{i-1} d_j$ columns, and the number of rows is reduced to $N - 1$. Next, it is necessary to populate the structures needed to discard columns from the new array $\mathcal{A}'$ with the reduced number of rows, and then, the number of columns is reduced to $k - d_i$.

Fig 6 shows an example of the application of $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$ to the problem instance presented in Fig 3. Because Δ is not greater than δ in this instance, the number of columns that must be eliminated with the elimination of each row is given by the vector $\mathcal{D} = \{d_1 = 1, d_2 = 0\}$; i.e., after the deletion of the first row, one column must be deleted, and then the algorithm proceeds to the deletion of the second row to satisfy the value $\delta = 2$. The example shown in Fig 6 illustrates the reduction process for the instance given in Fig 3. The first column lists the main structures that are changed during the execution of the algorithm. Each of the remaining columns in Fig 6 represents a different iteration of the main loop of the algorithm.

Algorithm 3

```
function GG_{RC}(A, δ, Δ)
    J_R = ∅
    J_C = ∅
    if Δ > 0 then
      if Δ > δ then
        for i ← 0 to i < δ − 1 do
          d_i ← ⌊Δ/δ⌋
        end for
        d_{δ−1} ← ⌈Δ/δ⌉
      else
        for i ← 0 to i < Δ do
          d_i ← 1
        end for
      end if
      for i ← 0 to i < δ do
          GETN(A')
          r ← F_R(O)
          A' ← A'_R − r
          GETK(A')
          for j ← 0 to j < d_i do
              c ← F_C(K, k, U)
              A' ← A'_C − c
          end for
      end for
      B ← A'
    else
      GG_{RC}(A, δ, Δ)
    end if
    return (B)
end function
```

Meta-heuristic algorithm $\mathcal{MM}$

The approximate algorithm $\mathcal{MM}$ for searching for a solution to the OSCAR problem is based on the SA approach and is described in Algorithm 4. This approach is a general-purpose stochastic optimization strategy that has been proven to be an efficient means of approximating global optimal solutions to many NP-complete combinatorial optimization problems. In this strategy, a solution $\mathcal{W}_u$ is first constructed using the `Initialize(...)` method, and this solution is designated as the first global best solution $\mathcal{W}^*$; then, the algorithm enters an iterative improvement process, controlled by the length of the Markov chain, until a certain termination criterion is achieved. In each iteration of this improvement process, a new solution $\mathcal{W}_v$ is generated using the `GenerateNeighbor(...)` method, and this new solution is substituted for $\mathcal{W}_u$ whenever its quality is superior to that of the current solution or the probability condition is satisfied. The probability condition is based on the Boltzmann distribution, and it is defined with respect to the values of an initial temperature $\mathcal{T}_i$, a final temperature $\mathcal{T}_f$, and a quality function $\tau(\ldots)$ of $\mathcal{W}_u$ and $\mathcal{W}_v$. The global best $\mathcal{W}^*$ is also updated every time a solution $\mathcal{W}_v$ improves upon it. The details of these procedures are presented in the remainder of this subsection.

Algorithm 4

```
function MM(A, δ, Δ, T_i, T_f, α, E, P)
  J_R = ∅
  J_C = ∅
  W_v ← Initialize(N, k, δ, Δ)
  W_u ← W_v
  W* ← W_v
  L ← (N + k) v
  n ← 0
  while n < E and T_i < T_f do
    for i ← 0 to i < L do
      W_v ← GenerateNeighbor (S, W_u, P)
      if τ(W_v) ≤ τ(W_u) then
        W_u ← W_v
        if τ(W_u) ≤ τ(W*) then
          W* ← W_u
        else if random(0...1) ≤ e^((τ(W_v) − τ(W_u)) / T_i) then
          W_u ← W_v
        end if
      end if
    end for
    T_i ← α * T_i
    n ← n + 1
  end while
  B ← constructMatrix (W*)
  return (B)
end function
```

A solution is represented by a vector $\mathcal{W}$ with $N + k$ elements, which identify the subsets of the rows and columns of the matrix $\mathcal{A}$ that are used to construct a new submatrix $\mathcal{B}$. All of the elements are binary; each of the first N elements is associated with a particular row in $\mathcal{A}$, and each of the last k elements is associated with one of its columns. The subset $\mathcal{J}_{\mathcal{R}} \subset \mathcal{A}_{\mathcal{R}}$ of rows to be excluded from the new submatrix $\mathcal{B}$ consists of the rows of $\mathcal{A}_{\mathcal{R}}$ that are associated with the corresponding elements in $\mathcal{W}$ that have a value of 1. Similarly, the subset $\mathcal{J}_{\mathcal{C}} \subset \mathcal{A}_{\mathcal{C}}$ of columns to be excluded from the new submatrix $\mathcal{B}$ consists of the columns of $\mathcal{A}_{\mathcal{C}}$ that are associated with the corresponding elements in $\mathcal{W}$ that have a value of 1.

The `Initialize(N, k, δ, Δ)` method is used to construct the first solution $\mathcal{W}_u$ in the proposed strategy. This method uses the best greedy algorithm among those proposed in this paper. The greedy strategy is chosen based on preliminary experiments for SA_{OSCAR}.

Once the initial solution $\mathcal{W}_u$ has been created, it is modified using the neighborhood function `GenerateNeighbor`$(\mathcal{S}, \mathcal{W}_u, \mathcal{P})$. This method randomly chooses from among three predefined strategies, $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$, to create a new solution $\mathcal{W}_v$. In the strategy $\mathcal{S}_1$, a row of $\mathcal{W}_u$ is exchanged; one existing row in the solution is randomly removed and replaced with a different row not previously included. The strategy $\mathcal{S}_2$ follows the same approach as that of $\mathcal{S}_1$ but for columns. Finally, the strategy $\mathcal{S}_3$ is the combination of the previous two. The use of the neighborhood function is controlled by the parameter L, which is called the Markov chain length.

The quality function, or evaluation function, that is used to measure the fitness of a solution is derived from the definition of the OSCAR problem. This function, denoted by $\tau(\mathcal{W})$, counts the number of missing t-wise combinations. We note that one missing t-wise combination represents a t-tuple for a particular combination of columns that the matrix does not contain but would be required to cover in order for the matrix to be a $CA(N - \delta; t, k - \Delta, v)$.

Finally, the cooling schedule is controlled by a cooling factor α, which is used to gradually decrease an initial temperature $\mathcal{T}_i$ until it reaches a given final temperature $\mathcal{T}_f$, marking the

end of the algorithm. Note that the algorithm also includes an alternative termination criterion, which is defined as a maximum number of iterations of the main loop.

The time required to execute $\mathcal{MM}$ is $O(iL)$, where $i = \frac{\log_\alpha \mathcal{T}_f - \log_\alpha \mathcal{T}_i}{\log_\alpha \alpha}$ is the number of temperature decrements necessary to reach $\mathcal{T}_f$.

Exact algorithms $\mathcal{EE}_{\mathcal{RC}}$ and $\mathcal{EE}_{\mathcal{CR}}$

The exact algorithms $\mathcal{EE}_{\mathcal{RC}}$ and $\mathcal{EE}_{\mathcal{CR}}$ were previously reported in [24]. They follow the B&B strategy (cf. [43]) and avoid the need to explore the entire feasible region to find the optimal solution. The general idea behind these algorithms is described in Algorithm 5 and Algorithm 6.

These previously presented algorithms first construct an initial solution $\mathcal{B}$ using a greedy strategy; then, they remove from this solution all possible combinations of rows and columns such that the resulting matrix $\mathcal{B}'$ has $N - \delta$ rows and $k - \Delta$ columns. Because the order in which the rows and columns are removed matters, the strategies differ in their selection of which elements are removed first. Whereas $\mathcal{EE}_{\mathcal{RC}}$ first removes a subset of columns, $\mathcal{EE}_{\mathcal{CR}}$ first removes a subset of rows. Both algorithms, after the first selected elements have been removed, perform a B&B search over the columns and/or rows, testing each element one by one, in order to find the submatrix $\mathcal{B}^*$ with the minimum number of missing t-wise combinations.

Algorithm 5

```
1: function EE_RC (A, δ, Δ)
2:     B* ← bestGreedy (A, δ, Δ)
3:     B ← B*
4:     UpperBound ← τ(B)
5:     for each J_C ⊂ A_C and |J_C| = Δ do
6:       B' ← Eliminate (B, J_C)
7:       for each J_R ⊂ A_R and |J_R| = δ do
8:         B'' ← Eliminate (B', J_R)
9:         if τ(B'') < UpperBound then
10:            B* ← B''
11:            UpperBound ← τ(B'')
12:        end if
13:      end for
14:    end for
15:    return (B*)
16: end function
```

Algorithm 6

```
1: function EE_CR(A, δ, Δ)
2:     B* ← bestGreedy (A, δ, Δ)
3:     B ← B*
4:     UpperBound ← τ(B)
5:     for each J_R ⊂ A_R and |J_R| = δ do
6:       B' ← Eliminate (B, J_R)
7:       for each J_C ⊂ A_C and |J_C| = Δ do
8:         B'' ← Eliminate (B', J_C)
9:         if τ(B'') < UpperBound then
10:            B* ← B''
11:            UpperBound ← τ(B'')
12:        end if
13:      end for
14:    end for
15:    return (B*)
16: end function
```

We note that the elimination of the second set of elements obeys a lexicographical order given by the columns or rows that are being deleted. Moreover, during the search process, both algorithms avoid the elimination of certain columns or rows that would exert undesirable effects on the final submatrix (i.e., selections for which the number of missing t-wise combinations would increase over a certain upper bound); for a more detailed description of the algorithms, refer to [24]. The time required to execute either $\mathcal{EE}_{\mathcal{CR}}$ or $\mathcal{EE}_{\mathcal{RC}}$ is $O\left(\binom{N}{N-\delta}\binom{k}{k-\Delta}\right)$ since, in the worst case, it is not possible to discard solutions.

Hybrid algorithms for solving the OSCAR problem

Hybrid algorithm $\mathcal{GE}$. The algorithm $\mathcal{GE}$ combines the greedy and exact approaches to solve the OSCAR problem. The algorithm proceeds in two phases. First, it chooses a set of Δ columns and removes them from the initial matrix $\mathcal{A}$; the resulting matrix is denoted by $\mathcal{A}'$ and has dimensions of $N \times (k - \Delta)$. Afterward, the algorithm discards δ rows from $\mathcal{A}'$ in a greedy manner to construct a possible solution $\mathcal{B}'$, i.e., a matrix with dimensions of $(N - \delta) \times (k - \Delta)$, for the OSCAR instance at hand. To obtain the best solution $\mathcal{B}$, the algorithm explores all possible combinations of Δ columns and identifies the best matrix $\mathcal{B}$ from among all matrices $\mathcal{B}'$ constructed during the process described above.

Algorithm 7

```
1: function GE (A, δ, Δ)
2:   for i ← 0 to i < C(k, k−Δ) do
3:       J_C ← GREATERTHANPOLYNOMIAL (J_C)
4:       A' ← A − J_C^c
5:       GETN (A')
6:       J_R = ∅
7:       for j ← 0 to j < δ do
8:           r ← F_R(O)
9:           J_R^c = J_R^c + r
10:      end for
11:      B' ← A'_R − J_R^c
12:      if τ(B') < τ(B) then
13:         B ← B'
14:         τ(B) ← τ(B')
15:      end if
16:   end for
17:   return (B)
18: end function
```

The algorithm $\mathcal{GE}$ for solving the OSCAR problem is described in Algorithm 7. Each combination of $k - \Delta$ columns is represented by the vector $\mathcal{J}_{\mathcal{C}}$. Each new combination of $k - \Delta$ columns is computed by the function `GREATERTHANPOLYNOMIAL()` [44]. An array $\mathcal{A}'$, with dimensions of $N \times (k - \Delta)$, is constructed using the columns indicated by $\mathcal{J}_{\mathcal{C}}$. Then, the algorithm populates the necessary structures to reduce $\mathcal{A}'$ to a matrix with $N - \delta$ rows, and this reduction process yields an array $\mathcal{B}'$. The best solution that has been found so far during the exploration process is represented by $\mathcal{B}$. Whenever $\tau(\mathcal{B}') < \tau(\mathcal{B})$ for a newly constructed matrix $\mathcal{B}'$, the matrix $\mathcal{B}$ is replaced with $\mathcal{B}'$.

The time required to execute $\mathcal{GE}$ can be derived from the times required for populating the necessary structures, reducing the number of rows, and updating the necessary values. This time is proportional to $O\left(\binom{k}{k-\Delta}\left(\left(N + v^t + v^t\binom{N}{2}\right)\binom{k-\Delta}{t} + \delta\left(N + (N-1)\binom{k-\Delta}{t}\right)\right)\right)$.

Hybrid algorithm $\mathcal{EG}$. The algorithm $\mathcal{EG}$ also combines the exact and greedy approaches to find a solution to the OSCAR problem; compared with $\mathcal{GE}$, the difference is that it explores all possible combinations of δ rows that can be eliminated from the original matrix. The

algorithm proceeds in two phases. First, it chooses a set of δ rows and removes them from the initial matrix $\mathcal{A}$; the resulting matrix is denoted by $\mathcal{A}'$ and has dimensions of $(N-\delta)\times k$. Subsequently, the algorithm greedily discards Δ columns from $\mathcal{A}'$ to construct a possible solution $\mathcal{B}'$, i.e., a matrix with dimensions of $(N-\delta)\times(k-\Delta)$, for the OSCAR instance at hand. To obtain the best solution $\mathcal{B}$, the algorithm explores all possible combinations of δ rows and identifies the best matrix $\mathcal{B}$ from among all matrices $\mathcal{B}'$ constructed during the process described above.

The algorithm $\mathcal{EG}$ for solving the OSCAR problem is described in Algorithm 8. Each combination of $N-\delta$ rows is represented by the vector $\mathcal{J}_{\mathcal{R}}$. Each new combination of $N-\delta$ rows is computed by the function `GREATERTHANPOLYNOMIAL()`. An array $\mathcal{A}'$, with dimensions of $(N-\delta)\times k$, is constructed using the rows indicated by $\mathcal{J}_{\mathcal{R}}$. Then, the algorithm populates the necessary structures to greedily reduce $\mathcal{A}'$ to a matrix with $k-\Delta$ columns, and this reduction process yields an array $\mathcal{B}'$. The best solution that has been found so far during the exploration process is represented by $\mathcal{B}$. Whenever $\tau(\mathcal{B}')<\tau(\mathcal{B})$ for a newly constructed matrix $\mathcal{B}'$, the matrix $\mathcal{B}$ is replaced with $\mathcal{B}'$.

Algorithm 8

```
1: function EG(A, δ, Δ)
2:   for i ← 0 to i < (N choose N−δ) do
3:       J_R ← GREATERTHANPOLYNOMIAL(J_R)
4:       A' ← A − J_R^c
5:       GETK(A')
6:       J_C = ∅
7:     for j ← 0 to j < Δ do
8:         c ← F_C(G, K)
9:         J_C^c ← J_C^c + c
10:    end for
11:    B' ← A_C' − J_C^c
12:    if τ(B') < τ(B) then
13:        B ← B'
14:        τ(B) ← τ(B')
15:    end if
16:  end for
17:  return (B)
18: end function
```

The time required to execute $\mathcal{EG}$ can be derived from the times required for populating the necessary structures, reducing the number of columns, and updating the necessary values. This time is proportional to $O\left(\binom{N}{N-\delta}\left(\left(N+\binom{t}{2}+t\right)\binom{k}{t}+\Delta(2k)\right)\right)$.

Hybrid algorithm $\mathcal{GM}$. The algorithm $\mathcal{GM}$ uses a hybrid strategy that combines the SA meta-heuristic [45] with the greedy approach to construct a solution to the OSCAR problem. In each iteration of $\mathcal{GM}$, a local search is performed over the possible set of columns that can be eliminated to obtain a matrix $\mathcal{A}'$ with dimensions of $N\times(k-\Delta)$. Afterward, the matrix $\mathcal{A}'$ is subjected to a greedy process to reduce its size by δ rows and thus to construct a solution $\mathcal{B}$ with dimensions of $(N-\delta)\times(k-\Delta)$ for the OSCAR instance at hand. Once the matrix $\mathcal{B}$ has been built, the Boltzmann criterion is used as usual in SA. The details of the strategy are presented in the remainder of this subsection.

Algorithm 9 describes the proposed $\mathcal{GM}$ approach for solving the OSCAR problem. The algorithm $\mathcal{GM}$ uses a vector W_u of size k to represent the state of each column in the solution $\mathcal{B}$. The elements of the vector take values of $w_i\in\{0,1\}$ for $1\leq i\leq k$, where a value of 0 indicates that the corresponding column is not present in the solution and a value of 1 indicates otherwise. In addition, two constraints are imposed to obtain a proper OSCAR solution: there

must be $k - \Delta$ elements with a value of $w_i = 1$, and there must be Δ elements with a value of $w_i = 0$.

Algorithm 9

```
1: function GM(A, δ, Δ, T_i, T_f, α, E)
2:    W_v ← INITIALIZE(k, Δ)
3:    W_u ← W_v
4:    W' ← W_v
5:    L ← (N + k) v
6:    n ← 0
7:   while n < E and T_i < T_f do
8:     for i ← 0 to i < L do
9:        W_v ← GENERATENEIGHBOR(W_u)
10:       A' ← CONSTRUCTSOLUTION
11:       GETN(A')
12:       for i ← 0 to i < δ do
13:          F_R(O)
14:       end for
15:       if τ(W_v) ≤ τ(W_u) then
16:          W_u ← W_v
17:         if τ(W_u) ≤ τ(W') then
18:            W' ← W_u
19:         end if
20:       else if RANDOM(0 ... 1) ≤ e^((τ(W_v)−τ(W_u))/T_i) then
21:         W_u ← W_v
22:       end if
23:     end for
24:     T_i ← α · T_i
25:   end while
26:   B ← CONSTRUCTMATRIX(W')
27:   return (B)
28: end function
```

The algorithm $\mathcal{GM}$ uses a set of perturbations to the vector $\mathcal{W}_u$ as its neighborhood function. For this purpose, it chooses two elements w_i and w_j, where $w_i \neq w_j$, and interchanges their values. The new solution formed via this perturbation, which is a neighbor of $\mathcal{W}_u$, is denoted by $\mathcal{W}_v$.

Finally, the evaluation function used in $\mathcal{GM}$ is τ, the number of missing combinations in a created matrix. This function is also used to evaluate the matrices created during the local search.

The time required to execute $\mathcal{GM}$ is proportional to $O(iLF_R)$, where $i = \frac{\log_\alpha \mathcal{T}_f - \log_\alpha \mathcal{T}_i}{\log_\alpha \alpha}$ is the number of temperature decrements necessary to reach $\mathcal{T}_f$ and F_R is the time cost of the greedy approach for eliminating rows.

Hybrid algorithm $\mathcal{MG}$. The algorithm $\mathcal{MG}$ uses another hybrid strategy that combines the SA meta-heuristic [45] with the greedy approach to construct a solution to the OSCAR problem. In each iteration of $\mathcal{MG}$, a local search is performed over the possible set of rows that can be eliminated to obtain a matrix $\mathcal{A}'$ with dimensions of $(N - \delta) \times k$. Afterward, the matrix $\mathcal{A}'$ is subjected to a greedy process to reduce its size by Δ columns and thus to construct a solution $\mathcal{B}$ with dimensions of $(N - \delta) \times (k - \Delta)$ for the OSCAR instance at hand. Once the matrix $\mathcal{B}$ has been built, the Boltzmann criterion is used as usual in SA. The details of the strategy are presented in the remainder of this subsection.

Algorithm 10 describes the proposed $\mathcal{MG}$ approach for solving the OSCAR problem. The algorithm $\mathcal{MG}$ uses a vector W_u of size N to represent the state of each row in the solution $\mathcal{B}$.

The elements of the vector take values of $w_i \in \{0, 1\}$ for $1 \leq i \leq k$, where a value of 0 indicates that the corresponding row is not present in the solution and a value of 1 indicates otherwise. The constraints imposed to ensure a proper OSCAR solution are as follows: there must be $N - \delta$ elements with a value of $w_i = 1$ and δ elements with a value of $w_i = 0$.

Algorithm 10

```
1: function MG(A_{N×k}, t, δ, Δ, T_i, T_f, α, E)
2:   W_v ← INITIALIZE(k, Δ)
3:   W_u ← W_v
4:   W' ← W_v
5:   L ← (N + k) v
6:   n ← 0
7:   while n < E and T_i < T_f do
8:     for i ← 0 to i < L do
9:       W_v ← GENERATENEIGHBOR(W_u)
10:      A' ← CONSTRUCTSOLUTION
11:      GETK (A', N, t, k, v)
12:      for i ← 0 to i < Δ do
13:        F_C(G, K)
14:      end for
15:      if τ(W_v) ≤ τ(W_u) then
16:        W_u ← W_v
17:        if τ(W_u) ≤ τ(W') then
18:          W' ← W_u
19:        end if
20:      else if RANDOM(0 ... 1) ≤ e^{(τ(W_v)−τ(W_u))/T_i} then
21:        W_u ← W_v
22:      end if
23:    end for
24:    T_i ← α · T_i
25:  end while
26:  B ← CONSTRUCTMATRIX(W')
27:  return (B)
28: end function
```

The algorithm $\mathcal{MG}$ uses a set of perturbations to the vector $\mathcal{W}_u$ as its neighborhood function. For this purpose, it chooses two elements w_i and w_j, where $w_i \neq w_j$, and interchanges their values. The new solution formed via this perturbation, which is a neighbor of $\mathcal{W}_u$, is denoted by $\mathcal{W}_v$.

Finally, the evaluation function used in $\mathcal{GM}$ is τ, the number of missing combinations in a created matrix. This function is also used to evaluate the matrices created during the local search.

The time required to execute $\mathcal{MG}$ is $O(iLF_C)$, where $i = \frac{\log_\alpha \mathcal{T}_f - \log_\alpha \mathcal{T}_i}{\log_\alpha \alpha}$ is the number of temperature decrements necessary to reach $\mathcal{T}_f$ and F_C is the time cost of the greedy approach for eliminating columns.

Hybrid algorithm $\mathcal{ME}$. The algorithm $\mathcal{ME}$ combines the meta-heuristic and exact approaches to solve the OSCAR problem, using a strategy based on the exploration of all possible combinations of Δ columns that can be eliminated from the original matrix. The algorithm proceeds in two phases. First, it chooses a set of Δ columns and removes them from the initial matrix $\mathcal{A}$; the resulting matrix is denoted by $\mathcal{A}'$ and has dimensions of $N \times (k - \Delta)$. Then, the algorithm uses the SA approach to discard δ rows from $\mathcal{A}'$ to construct a possible solution $\mathcal{B}'$, i.e., a matrix with dimensions of $(N - \delta) \times (k - \Delta)$, for the OSCAR instance at hand. To obtain the best solution $\mathcal{B}$, the algorithm explores all possible combinations of

Δ columns and identifies the best matrix $\mathcal{B}$ from among all matrices $\mathcal{B}'$ constructed during the process described above.

Algorithm 11 describes our $\mathcal{ME}$ approach. For each possible combination of columns $\mathcal{J}_\mathcal{C} \subset \mathcal{A}_\mathcal{C}$, the algorithm performs a meta-heuristic search to define the set of rows $\mathcal{J}_\mathcal{R} \subset \mathcal{A}_\mathcal{R}$. To determine each element in $\mathcal{J}_\mathcal{C}$, the function `GREATERTHANPOLYNOMIAL()` [44] is used to systematically generate each different combination of columns.

Algorithm 11

```
1: function ME(A, δ, Δ, T_i, T_f, α, E)
2:   for i ← 0 to i < C(k, k−Δ) do
3:     J_C ← GREATERTHANPOLYNOMIAL(J_C)
4:     W_v ← INITIALIZE(N, δ)
5:     W_u ← W_v
6:     W' ← W_v
7:     L ← (N + k) v
8:     n ← 0
9:     while n < E and T_i < T_f do
10:       for j ← 0 to j < L do
11:         W_v ← GENERATENEIGHBOR(W_u)
12:         if τ(W_v) ≤ τ(W_u) then
13:           W_u ← W_v
14:           if τ(W_u) ≤ τ(W') then
15:             W' ← W_u
16:           end if
17:         else if RANDOM(0 ... 1) ≤ e^((τ(W_v)−τ(W_u))/T_i) then
18:           W_u ← W_v
19:         end if
20:       end for
21:       T_j ← α · T_j
22:     end while
23:     if τ(W') < bestS) then
24:       bestS ← W'
25:       B ← constructMatrix (J_C, W')
26:     end if
27:   end for
28:   return (B)
29: end function
```

The time required to execute $\mathcal{ME}$ is $O\left(\binom{k}{k-\Delta} iL\right)$, where $i = \frac{\log_\alpha \mathcal{T}_f - \log_\alpha \mathcal{T}_i}{\log_\alpha \alpha}$ is the number of temperature decrements necessary to reach $\mathcal{T}_f$.

Hybrid algorithm $\mathcal{EM}$. The algorithm $\mathcal{EM}$ also combines the meta-heuristic and exact approaches to solve the OSCAR problem; compared with $\mathcal{ME}$, the difference is that it explores all possible combinations of δ rows that can be eliminated from the original matrix. The algorithm proceeds in two phases. First, it chooses a set of δ rows and removes them from the initial matrix $\mathcal{A}$; the resulting matrix is denoted by $\mathcal{A}'$ and has dimensions of $(N - \delta) \times k$. Then, the algorithm uses the SA approach to discard Δ columns from $\mathcal{A}'$ to construct a possible solution $\mathcal{B}'$, i.e., a matrix with dimensions of $(N - \delta) \times (k - \Delta)$, for the OSCAR instance at hand. To obtain the best solution $\mathcal{B}$, the algorithm explores all possible combinations of δ rows and identifies the best matrix $\mathcal{B}$ from among all matrices $\mathcal{B}'$ constructed during the process described above.

Algorithm 12 describes our $\mathcal{EM}$ approach. For each possible combination of rows $\mathcal{J}_\mathcal{R} \subset \mathcal{A}_\mathcal{R}$, the algorithm performs a meta-heuristic search to define the set of columns

$\mathcal{J}_{\mathcal{C}} \subset \mathcal{A}_{\mathcal{C}}$. To determine each element in $\mathcal{J}_{\mathcal{R}}$, the function GRATERTHANPOLYNOMIAL() [44] is used to systematically generate each different combination of rows.

Algorithm 12

```
1: function EM(A, δ, Δ, T_i, T_f, α, E)
2:   for i ← 0 to i < C(k, k−δ) do
3:     J_R ← GREATERTHANPOLYNOMIAL(J_R)
4:     W_v ← INITIALIZE(k, Δ)
5:     W_u ← W_v
6:     W' ← W_v
7:     L ← (N + k)v
8:     n ← 0
9:     while n < E and T_i < T_f do
10:       for i ← 0 to i < L do
11:         W_v ← GENERATENEIGHBOR(W_u)
12:         if τ(W_v) ≤ τ(W_u) then
13:           W_u ← W_v
14:           if τ(W_u) ≤ τ(W') then
15:             W' ← W_u
16:           end if
17:         else if RANDOM(0 ... 1) ≤ e^((τ(W_v)−τ(W_u))/T_i) then
18:           W_u ← W_v
19:         end if
20:       end for
21:       T_i ← α · T_i
22:     end while
23:     if τ(W') < bestS) then
24:       bestS ← W'
25:       B ← constructMatrix(J_R, W')
26:     end if
27:   end for
28:   return (B)
29: end function
```

The time required to execute $\mathcal{EM}$ is $O\left(\binom{N}{N-\Delta} iL\right)$, where $i = \frac{\log_{\alpha} \mathcal{T}_f - \log_{\alpha} \mathcal{T}_i}{\log_{\alpha} \alpha}$ is the number of temperature decrements necessary to reach $\mathcal{T}_f$.

In the next section, we demonstrate the performance of our 12 algorithms.

Experimentation

This section presents the experimental design used to test the performance of the proposed algorithms for solving the OSCAR problem. The methodology consisted of the following steps: 1) A set of benchmark instances was defined. 2) The parameters of the SA algorithm were subjected to a fine-tuning process. 3) The performances of the algorithms were evaluated by using them to solve the benchmark problem instances. 4) A performance comparison against state-of-the-art initialization functions was conducted. 5) The results derived from the algorithms were used to define new upper bounds for existing CAs.

The proposed algorithms were implemented in the `C` language and compiled using `gcc` with the optimization option `-O3`. We used a computer with 72 `Intel Xeon` 1.6 GHz CPU cores and RAM of 64 GB. The remainder of this section describes the experimental methodology in detail.

Table 13. Benchmark $\mathcal{L}_1$, which is composed of 12 small instances of the OSCAR problem. The column 2 shows the CA uses as initial array, while the columns 3 and 4 show the number of rows δ and columns Δ to be shortened, respectively.

Instance	$\mathcal{A}$	δ	Δ
1	CA(188;2,140,9)	3	0
2	CA(194;2,36,10)	2	0
3	CA(206;2,78,10)	1	4
4	CA(165;2,14,12)	1	1
5	CA(247;2,18,14)	3	1
6	CA(255;2,18,15)	3	1
7	CA(355;2,12,18)	1	4
8	CA(498;2,29,18)	1	1
9	CA(511;2,22,20)	1	2
10	CA(511;2,22,20)	2	3
11	CA(520;2,22,21)	1	2
12	CA(520;2,22,21)	2	4

https://doi.org/10.1371/journal.pone.0189283.t013

Definition of the benchmarks

This subsection introduces the three benchmarks used to properly test the proposed set of OSCAR algorithms. The benchmark $\mathcal{L}_1$ (S1 dataset) consists of 12 small CAs, which are described in Table 13, and it is used to analyze the performance of all algorithms presented in this document; then, the algorithms that achieve the best experimental results on this benchmark in terms of both time and solution quality are further tested on the following benchmark. The benchmark $\mathcal{L}_2$ (S2 dataset), presented in Table 14, consists of 62 CAs; it is an extension of the benchmark presented in [24] such that the adjusted values of δ and Δ provide support for the discovery of a greater number of new upper bounds for the related CAs. This benchmark aids in the identification of the OSCAR solver with the best overall experimental performance, and it is also used to compare the results of the proposed OSCAR solvers against other state-of-the art initialization functions. Finally, the benchmark $\mathcal{L}_3$ (S3 dataset) consists of 820 instances (see Table 15); this benchmark is used to evaluate the quasi-CA construction performance of IPOG-F, a classical and versatile (in the sense that it can rapidly construct any type of CA) greedy algorithm that is widely used in the literature, against the best OSCAR strategies identified in the experiments on the previous benchmarks in terms of both the time required for matrix construction and the quality of the constructed matrices. Table 15 presents the instances included in benchmark $\mathcal{L}_3$, organized into 20 sets. In each set, one OSCAR instance is defined per value of k considered (from 10 to 50), as shown in column 1; the remaining columns show the values for v, t, δ, and Δ, which correspond to the alphabet size, the strength, and the numbers of rows and columns to be eliminated, respectively. We note that the benchmark $\mathcal{L}_3$ is also characterized by its wide variety of values of the strength t and the alphabet size v.

Fine-tuning of the parameters of $\mathcal{MM}$

The $\mathcal{MM}$ approach is the basis for several of our other approaches. Because this approach uses the SA algorithm, a fine-tuning process is necessary to adjust the values of its parameters to improve its performance. During the tuning process performed in this study, the Markov chain length L, the final temperature $\mathcal{T}_f$, and the initialization function $\mathcal{G}$ were fixed; all remaining parameters (i.e., the initial temperature $\mathcal{T}_i$, the decrement factor α, and the maximum number of evaluations $\mathcal{E}$) were subjected to adjustment. Because different neighborhood

Table 14. Benchmark $\mathcal{L}_2$, which is composed of 62 instances of the OSCAR problem. Each instance shows the initial array $\mathcal{A}$, and the number of rows δ and columns Δ to be shortened.

Instance	$\mathcal{A}$	δ	Δ	Instance	$\mathcal{A}$	δ	Δ
1	CA(53;2,52,5)	4	9	32	CA(511;2,22,20)	1	2
2	CA(53;2,52,5)	3	7	33	CA(511;2,22,20)	2	3
3	CA(53;2,52,5)	2	5	34	CA(511;2,22,20)	3	4
4	CA(93;2,113,6)	1	6	35	CA(511;2,22,20)	4	6
5	CA(188;2,140,9)	3	0	36	CA(511;2,22,20)	9	7
6	CA(120;2,80,8)	1	51	37	CA(511;2,22,20)	10	8
7	CA(120;2,80,8)	2	52	38	CA(511;2,22,20)	12	9
8	CA(153;2,99,9)	2	69	39	CA(511;2,22,20)	14	10
9	CA(194;2,36,10)	2	0	40	CA(511;2,22,20)	16	11
10	CA(206;2,78,10)	1	4	41	CA(511;2,22,20)	20	12
11	CA(165;2,14,12)	1	1	42	CA(511;2,22,20)	29	13
12	CA(165;2,14,12)	2	4	43	CA(511;2,22,20)	46	14
13	CA(247;2,18,14)	3	1	44	CA(511;2,22,20)	82	15
14	CA(247;2,18,14)	4	2	45	CA(520;2,22,21)	1	2
15	CA(247;2,18,14)	5	3	46	CA(520;2,22,21)	2	4
16	CA(247;2,18,14)	6	4	47	CA(520;2,22,21)	3	6
17	CA(247;2,18,14)	7	5	48	CA(520;2,22,21)	4	8
18	CA(247;2,18,14)	8	6	49	CA(520;2,22,21)	6	9
19	CA(247;2,18,14)	9	7	50	CA(520;2,22,21)	7	11
20	CA(247;2,18,14)	11	8	51	CA(520;2,22,21)	19	13
21	CA(247;2,18,14)	14	9	52	CA(520;2,22,21)	21	14
22	CA(247;2,18,14)	18	10	53	CA(526;2,24,22)	1	3
23	CA(255;2,18,15)	3	1	54	CA(526;2,24,22)	2	8
24	CA(255;2,18,15)	4	4	55	CA(526;2,24,22)	3	12
25	CA(255;2,18,15)	5	7	56	CA(526;2,24,22)	4	13
26	CA(255;2,18,15)	6	8	57	CA(526;2,24,22)	5	14
27	CA(255;2,18,15)	7	10	58	CA(526;2,24,22)	6	16
28	CA(255;2,18,15)	9	11	59	CA(622;2,26,24)	1	4
29	CA(358;2,20,18)	3	8	60	CA(622;2,26,24)	2	10
30	CA(355;2,12,18)	1	4	61	CA(622;2,26,24)	3	16
31	CA(498;2,29,18)	1	1	62	CA(136;5,68,2)	2	33

https://doi.org/10.1371/journal.pone.0189283.t014

functions are used in our approach, each with a certain probability of being applied, a fourth parameter was also considered during the tuning process: the application probability of each neighbor function, denoted by $\mathcal{P}$. The goal of this fine-tuning process was to test the performance of $\mathcal{MM}$ using different configurations of the parameter values to identify the configuration that yielded the best performance.

The sets of values considered for the parameters $\mathcal{T}_i$, α, and $\mathcal{E}$ were {1, 4}, {0.90, 0.99}, and {100L, 500L}, respectively. In the fine-tuning approach presented in [46, 47], a CA is used as a means of systematically sampling the entire set of parameter value combinations; the method starts at an initial level of interaction t, which is used to construct a $CA(N; t, k, v)$, and t is then increased until the generated sample is suitable for the purposes of the experiment. The present study required the smallest possible sample in order to reduce the experimental time; this sample was constructed using an interaction level of $t = 2$. A summary of the final combinations of values tested, derived from the constructed $CA(4; 2, 3, 2)$, is shown in Table 16.

Table 15. Groups of instances' sets that form the benchmark $\mathcal{L}_3$. The column 1 is the identifier of the groups. The columng 2 shows the ranges of k, the number of columns. The remaining columns are the alphabet v, strength t, and rows δ and columns Δ to be shortened.

Instance set	$\mathcal{A}$	v	t	δ	Δ
1	$10 \leq k \leq 50$	2	2	1	1
2	$10 \leq k \leq 50$	2	3	1	1
3	$10 \leq k \leq 50$	2	4	1	1
4	$10 \leq k \leq 50$	2	5	1	1
5	$10 \leq k \leq 50$	3	2	1	1
6	$10 \leq k \leq 50$	3	3	1	1
7	$10 \leq k \leq 50$	3	4	1	1
8	$10 \leq k \leq 50$	4	2	1	1
9	$10 \leq k \leq 50$	4	3	1	1
10	$10 \leq k \leq 50$	5	2	1	1
11	$10 \leq k \leq 50$	5	3	1	1
12	$10 \leq k \leq 50$	6	2	1	1
13	$10 \leq k \leq 50$	6	3	1	1
14	$10 \leq k \leq 50$	2	2	1	0
15	$10 \leq k \leq 50$	2	3	1	0
16	$10 \leq k \leq 50$	2	4	1	0
17	$10 \leq k \leq 50$	2	5	1	0
18	$10 \leq k \leq 50$	3	2	1	0
19	$10 \leq k \leq 50$	3	3	1	0
20	$10 \leq k \leq 50$	3	4	1	0
21	$10 \leq k \leq 50$	4	2	1	0
22	$10 \leq k \leq 50$	4	3	1	0
23	$10 \leq k \leq 50$	5	2	1	0
24	$10 \leq k \leq 50$	5	3	1	0
25	$10 \leq k \leq 50$	6	2	1	0
26	$10 \leq k \leq 50$	6	3	1	0

https://doi.org/10.1371/journal.pone.0189283.t015

Meanwhile, the vector of probabilities $\mathcal{P}$ used for the initialization functions $\mathcal{S}_i$ was defined based on solutions to the Diophantine equation $a_1x + a_2x + a_3x = 10$, following the approach presented in [44]. During this process, each of the 66 solutions to the Diophantine equation was used to generate a possible vector $\mathcal{P}$, in which the probability value for each initialization function i was estimated as $\frac{x_i}{10}$.

Because we considered 4 different configurations of the values of the parameters $\mathcal{T}_i$, α, and $\mathcal{E}$ and 66 configurations of the probability vector $\mathcal{P}$, the experiment to fine-tune $\mathcal{MM}$

Table 16. Different parameter configurations that were tested for the algorithm $\mathcal{MM}$.

Code	$\mathcal{T}_i$	α	$\mathcal{E}$
C1	1	0.90	100L
C2	4	0.99	100L
C3	4	0.90	500L
C4	1	0.99	500L

https://doi.org/10.1371/journal.pone.0189283.t016

Table 17. Results obtained when solving $\mathcal{L}_1$ using the algorithms $\mathcal{GG}_{\mathcal{CR}}$, $\mathcal{GG}_{\mathcal{RC}}$, and $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$.

Quality of solution $\tau(\mathcal{B})$				Time (sec.)			
Instance	$\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$	$\mathcal{GE}$	$\mathcal{EG}$	Instance	$\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$	$\mathcal{GE}$	$\mathcal{EG}$
1	110	110	110	1	0.5	0.5	0.5
2	2	2	2	2	0.1	0.1	0.1
3	85	74	74	3	0.2	1.2	0.3
4	43	43	43	4	0.1	0.1	0.1
5	193	187	191	5	0.2	0.3	0.1
6	268	267	267	6	0.1	0.3	0.3
7	12	9	9	7	0.1	0.2	0.1
8	95	92	92	8	0.2	0.5	0.2
9	87	82	82	9	0.2	0.3	0.2
10	153	143	142	10	0.2	0.5	0.2
11	109	108	108	11	0.2	0.3	0.2
12	168	161	161	12	0.2	0.5	0.3

https://doi.org/10.1371/journal.pone.0189283.t017

involved 264 different parameter value configurations. Each configuration was used to solve two instances of the OSCAR problem, specified by $(\mathcal{A} = CA(31; 2, 35, 4), \delta = 5, \Delta = 5)$ and $(\mathcal{A} = CA(255; 2, 18, 15), \delta = 6, \Delta = 8)$, with a total of 31 runs per instance, where the value of the solution reported was the best among all the runs.

The results obtained from the fine-tuning process indicated that the optimal parameter values for $\mathcal{MM}$ are $\mathcal{T}_i = 4$, $\alpha = .99$, and $\mathcal{E} = 100L$ and that the desired solution to the Diophantine equation is $a_1 = 4$, $a_2 = 3$, and $a_3 = 3$. This configuration was also used in the algorithms $\mathcal{GM}$, $\mathcal{MG}$, $\mathcal{ME}$, and $\mathcal{EM}$, which also use the meta-heuristic $\mathcal{MM}$ approach.

Evaluation of the 12 proposed algorithms

This section presents the evaluation of the 12 proposed algorithms for solving the OSCAR problem. All algorithms were tested on the smaller set of 12 instances, $\mathcal{L}_1$, to identify the three best algorithms. Then, the larger set $\mathcal{L}_2$ was solved using only those three algorithms to further evaluate the general performance of these approaches.

The algorithms were first tested using the benchmark consisting of 12 OSCAR instances derived from 10 CAs taken from the literature. The values δ and Δ for these instances were fixed such that the size of the resulting array $\mathcal{B}$ would represent a possible new upper bound. In addition, these instances were created such that the size of the search space would permit us to solve them using all 12 algorithms; this was a concern because exact algorithms must explore all possible combinations of rows and columns, and therefore, if the search space is too large, they may require an excessive amount of time.

Table 17 presents the results obtained using each algorithm based solely on the greedy approach (i.e., the algorithms $\mathcal{GG}_{\mathcal{CR}}$, $\mathcal{GG}_{\mathcal{RC}}$, and $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$) when solving the benchmark $\mathcal{L}_1$. Note that $\mathcal{GG}_{\mathcal{RC}}$ and $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$ show better performance than $\mathcal{GG}_{\mathcal{CR}}$; when all instances are considered, the former algorithms result in equal or fewer missing t-wise combinations compared with the latter. Therefore, the findings show that it is beneficial to eliminate rows before columns (as in $\mathcal{GG}_{\mathcal{RC}}$ and $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$) when working with initial matrices that are already CAs. This is because when rows are removed from the matrix, those that contribute the least to the CA are chosen for deletion, and the t-wise combinations that are lost as a result can subsequently be compensated for by eliminating the columns that produce them. Meanwhile, although the time performance

Table 18. Results obtained when solving $\mathcal{L}_1$ using the algorithms $\mathcal{GE}$, $\mathcal{EG}$, and $\mathcal{GM}$.

Quality of solution $\tau(\mathcal{B})$				Time (sec.)			
Instance	$\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$	$\mathcal{GE}$	$\mathcal{EG}$	Instance	$\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$	$\mathcal{GE}$	$\mathcal{EG}$
1	110	110	110	1	0.5	41301.2	2507.3
2	2	2	2	2	0.2	0.2	0.2
3	74	74	74	3	97742.5	2.7	273.2
4	43	43	43	4	0.1	0.1	9225.3
5	187	187	187	5	0.2	1731.2	9286.5
6	267	267	267	6	0.2	2035.7	12507.5
7	9	9	9	7	2.3	0.2	23.5
8	105	105	105	8	2.5	2.7	927.5
9	82	82	82	9	10.8	1.2	867.5
10	142	142	142	10	78.5	268.2	608.2
11	108	108	108	11	15.3	1.5	1123.5
12	161	161	161	12	502.8	280.3	837.3

https://doi.org/10.1371/journal.pone.0189283.t018

of $\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$ is superior to that of the others for this particular set of instances, it will worsen rapidly with increasing values of δ and Δ. In general, the time performance of $\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$ will be the worst among the three greedy approaches, as indicated by the theoretical complexities presented alongside the definitions of these algorithms, mainly because of the greater number of calls to the greedy strategies for eliminating rows and columns.

Table 18 shows the results obtained using the hybrid approaches that combine the greedy strategy with either the exact approach or the meta-heuristic approach (i.e., the algorithms $\mathcal{GE}$, $\mathcal{EG}$, and $\mathcal{GM}$) when solving the benchmark $\mathcal{L}_1$. An increase in running time is observed for these approaches, mainly due to the use of the more elaborate strategies of the exact and meta-heuristic algorithms. However, the results achieved also improve upon some of the results obtained by the solely greedy algorithms. All of these hybrid algorithms achieve the same results; however, the average time increase for $\mathcal{GM}$ is much greater than that for the algorithms that include exact strategies. Let's point out that the small amounts of times appearing in the exact approach are indeed a result from its expected theoretical behavior.

Table 19 shows the results for another set of hybrid approaches, all involving the meta-heuristic strategy in combination with either the exact approach or the greedy approach (i.e., the

Table 19. Results obtained when solving $\mathcal{L}_1$ using the algorithms $\mathcal{MG}$, $\mathcal{ME}$, and $\mathcal{EM}$.

Quality of solution $\tau(\mathcal{B})$				Time (sec.)			
Instance	$\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$	$\mathcal{GE}$	$\mathcal{EG}$	Instance	$\mathcal{GG}^{\mathcal{R}}_{\mathcal{C}}$	$\mathcal{GE}$	$\mathcal{EG}$
1	110	110	110	1	5051.3	3252.3	15357.2
2	2	2	2	2	11300.8	2352.5	7542.5
3	94	94	94	3	47.2	5472.3	15.2
4	63	63	43	4	1399.7	0.2	11615.2
5	225	187	187	5	9850.2	9242.7	12503.2
6	308	267	267	6	12778.2	2582.3	15253.2
7	21	21	21	7	2.7	5.2	47.5
8	105	105	105	8	5082.5	1.2	5.3
9	93	82	82	9	35.2	15.2	82.5
10	181	181	181	10	35.7	17.2	42.7
11	108	108	108	11	12323.7	1.7	15.2
12	184	184	184	12	32.5	48.3	82.5

https://doi.org/10.1371/journal.pone.0189283.t019

Table 20. Results obtained when solving $\mathcal{L}_1$ using the algorithms $\mathcal{MM}$, $\mathcal{EE}_{\mathcal{CR}}$, and $\mathcal{EE}_{\mathcal{RC}}$.

Quality of solution $\tau(\mathcal{B})$				Time (sec.)			
Instance	$\mathcal{GG}_{c}^{\mathcal{R}}$	$\mathcal{GE}$	$\mathcal{EG}$	Instance	$\mathcal{GG}_{c}^{\mathcal{R}}$	$\mathcal{GE}$	$\mathcal{EG}$
1	110	110	110	1	0.3	32.8	3.7
2	2	2	2	2	0.3	0.2	0.2
3	74	74	74	3	0.3	12142.7	1252.3
4	43	43	43	4	0.5	0.1	1.2
5	187	187	187	5	1.5	1732.3	15242.2
6	267	267	267	6	5.7	1050.5	12345.2
7	9	9	9	7	0.3	135.2	15.5
8	105	105	105	8	0.3	8.5	82.3
9	82	82	82	9	0.3	0.5	5.2
10	142	142	142	10	0.3	23247.3	207872.5
11	108	108	108	11	1.2	7.2	63.5
12	161	161	161	12	323.2	232829.8	697482.5

https://doi.org/10.1371/journal.pone.0189283.t020

algorithms $\mathcal{MG}$, $\mathcal{ME}$, and $\mathcal{EM}$), when solving the benchmark $\mathcal{L}_1$. From these results and the previous ones shown in Table 18 for $\mathcal{GM}$, it can be seen that the algorithms $\mathcal{GM}$, $\mathcal{MG}$, $\mathcal{ME}$, and $\mathcal{EM}$ all find solutions with a comparable number of missing t-wise combinations to those in the solutions created by the other (exact or greedy) approaches. However, it should be noted that the initial matrices used in these algorithms were the best solutions obtained by a greedy algorithm, and in most cases, the differences in the number of missing t-wise combinations between these initial matrices and the results reported by the hybrid algorithms are nearly zero. These findings indicate that the contribution of these hybrid approaches is minimal.

Table 20 shows the results of solving the benchmark $\mathcal{L}_1$ using the algorithms $\mathcal{MM}$, $\mathcal{EE}_{\mathcal{CR}}$, and $\mathcal{EE}_{\mathcal{RC}}$. Note that $\mathcal{EE}_{\mathcal{CR}}$ exhibits better performance than $\mathcal{EE}_{\mathcal{RC}}$ because there are more possible ways to select rows than columns. However, exhaustive search algorithms are impractical for finding a solution to an OSCAR instance except when the values of δ and Δ are both quite small. The exact algorithm $\mathcal{EE}_{\mathcal{CR}}$ is more suitable when $N - \delta > k - \Delta$ since a greater portion of the search space is defined by $N - \delta$; otherwise, $\mathcal{EE}_{\mathcal{RC}}$ behaves better. However, the execution time of the proposed exact algorithms grows with the desired degree of reduction for a given instance, and they can become infeasible.

Finally, some additional important observations are noted in the following. First, for every instance, the solutions obtained by the algorithms $\mathcal{GE}$ and $\mathcal{EG}$ have the same number of missing t-wise combinations. When $N - \delta > k - \Delta$, $\mathcal{EG}$ requires more time than $\mathcal{GE}$; similarly, when $N - \delta < k - \Delta$, $\mathcal{GE}$ requires more time than $\mathcal{EG}$. These findings suggest that $\mathcal{GE}$ is appropriate when $N - \delta > k - \Delta$ and that $\mathcal{EG}$ is appropriate when $N - \delta < k - \Delta$.

Second, as δ and Δ increase for a given array $\mathcal{A}$, the number of missing t-wise combinations produced by the pure greedy algorithms increases in comparison with the results of the hybrid algorithms that include exact strategies, i.e., $\mathcal{EG}$ and $\mathcal{GE}$. For example, for the instances with the input array $CA(255; 2, 18, 15)$, we note that the solution obtained by $\mathcal{GG}_{c}^{\mathcal{R}}$ when $\delta = 3$ and $\Delta = 1$ has 267 missing t-wise combinations, whereas the solution obtained by $\mathcal{GE}$ has 257 missing t-wise combinations; similarly, when $\delta = 9$ and $\Delta = 11$, the solution obtained by $\mathcal{GG}_{c}^{\mathcal{R}}$ has 65 missing t-wise combinations, whereas the solution obtained by $\mathcal{GE}$ has 61 missing t-wise combinations. It can be inferred that the inclusion of an exact strategy contributes to reducing

the number of missing t-wise combinations, at the cost of an increase in the time required to build the matrix.

Third, and most importantly, the algorithms that showed the best performance in the experiment were $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$, $\mathcal{GE}$ and $\mathcal{EG}$; all of them obtained comparable solutions, with only small differences in both quality and time cost. The algorithms that include meta-heuristic strategies consumed considerably more time but showed little difference in the quality of their solutions, whereas the exact approaches are too expensive for large values of δ and Δ.

To further evaluate the proposed approaches, the algorithms $\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$, $\mathcal{GE}$ and $\mathcal{EG}$ were used to solve the benchmark $\mathcal{L}_2$, which includes larger CAs. Table 21 summarizes the results obtained when solving $\mathcal{L}_2$. In addition to this experiment, an instance specified by the array $\mathcal{A} = CA(136; 5, 68, 2)$ and values of $\delta = 2$ and $\Delta = 33$ was also solved using the meta-heuristic algorithm $\mathcal{MM}$; this algorithm produced a solution $\mathcal{B}$ with zero missing t-wise combinations, meaning that the approach constructed a new CA of the form $CA(134; 5, 35, 2)$. This last result serves as evidence that an approach based on seeking a solution to the OSCAR problem can also be used to construct CAs.

Performance comparison with state-of-the-art initialization algorithms

This subsection evaluates the performance of the proposed OSCAR approaches against the performance of several state-of-the-art initialization functions. For this purpose, the initialization functions described in the related work section are considered, and their results are compared with the best solutions obtained using the approaches proposed in this work.

The performance comparison was performed as follows. The benchmark $\mathcal{L}_2$ was chosen as the set of instances to be used in this evaluation. First, the OSCAR algorithms proposed in this work were used to solve the benchmark, and the best matrix $\mathcal{B}$ among all of the results was obtained for each instance. Then, the initialization functions, denoted by $\mathcal{I}_i$, were used to construct arrays $\mathcal{S}_i$ of the same dimensions as the matrices derived by solving the OSCAR instances; i.e., for each instance, we constructed an array $\mathcal{S}$ with $N - \delta$ rows and $k - \Delta$ columns. Once all of the solutions generated by the OSCAR algorithms and the state-of-the-art initialization functions had been obtained, they were evaluated with regard to the function τ, i.e., the number of missing t-wise combinations in each newly constructed matrix. Table 22 summarizes the results of this experiment. Column one shows the identifier of each instance in $\mathcal{L}_2$, column two shows the number of missing t-wise combinations in the best solution obtained using the OSCAR approaches, and columns three to six present the numbers of missing t-wise combinations in the solutions derived using the state-of-the-art initialization functions $\mathcal{I}_i$. Note that for the last problem instance, one of the proposed OSCAR approaches (the meta-heuristic algorithm $\mathcal{MM}$) was able to construct a CA, as seen from the fact that the new matrix has zero missing t-wise combinations.

Performance comparison with IPOG-F, a state-of-the-art CA construction approach

The experiment presented here involves the comparison of the $\mathcal{GG}_{\mathcal{RC}}$ and $\mathcal{EG}$ strategies against the state-of-the-art IPOG-F algorithm for CA construction. The goal in this experiment was to evaluate the performance when constructing CAs and/or quasi-CAs using IPOG-F, a fast greedy algorithm for CA construction that is widely used in the literature and is versatile in the sense that it can rapidly construct any type of CA. The $\mathcal{GG}_{\mathcal{RC}}$ and $\mathcal{EG}$ strategies are among the best of the proposed OSCAR solvers, as indicated by the experiments on the previous benchmarks. Both of these strategies were compared against IPOG-F in terms of the matrix

Table 21. Results obtained when solving $\mathcal{L}_2$ using the algorithms $\mathcal{GG}_{\mathcal{R}_c}$, $\mathcal{GE}$ and $\mathcal{EG}$.

Quality of solution $\tau(\mathcal{B})$				Time (sec.)			
Instance	$\mathcal{GG}_{\mathcal{R}_c}$	$\mathcal{GE}$	$\mathcal{EG}$	Instance	$\mathcal{GG}_{\mathcal{R}_c}$	$\mathcal{GE}$	$\mathcal{EG}$
1	272	-	267	1	0.1	-	349.7
2	229	-	216	2	0.1	-	32.5
3	173	172	172	3	0.1	1070.3	1.9
4	154	-	154	4	0.2	-	1.2
6	33	-	33	6	0.2	-	0.7
7	60	-	60	7	0.2	-	28.9
8	60	-	60	8	0.3	-	7357.2
12	41	40	40	12	0.2	1.5	3.5
14	214	214	214	14	0.3	0.2	100350.2
15	227	227	-	15	0.2	5.2	-
16	228	226	-	16	0.2	14.2	-
17	219	215	-	17	0.2	37.2	-
18	199	199	-	18	0.1	65.2	-
19	180	175	-	19	0.1	93.5	-
20	177	168	-	20	0.1	108.3	-
21	168	162	-	21	0.2	100.7	-
22	163	155	-	22	0.1	74.7	-
24	216	215	-	24	0.1	15.7	-
25	142	139	-	25	0.2	102.3	-
26	130	127	-	26	0.1	116.7	-
27	79	74	-	27	0.1	77.5	-
28	65	61	-	28	0.1	45.2	-
29	99	98	98	29	0.1	1088.2	7158.7
34	188	186	-	34	0.3	367.8	-
35	182	174	-	35	0.5	2433.5	-
36	370	363	-	36	0.5	5299.7	-
37	343	336	-	37	0.5	8503.7	-
38	346	337	-	38	0.5	15335.2	-
39	335	320	-	39	0.8	15820.8	-
40	309	292	-	40	0.8	13832.8	-
41	301	286	-	41	0.8	9502.3	-
42	346	339	-	42	0.8	8112.7	-
43	455	434	-	43	1.2	3703.5	-
44	681	641	-	44	1.2	2144.2	-
47	180	173	173	47	0.2	4003.2	45270.3
48	161	161	-	48	0.5	11791.7	-
49	208	201	-	49	0.7	16457.2	-
50	154	140	-	50	0.7	15723.8	-
51	275	263	-	51	1.2	8457.2	-
52	235	212	-	52	1.5	4220.5	-
53	140	140	140	53	0.3	242.7	1.7
54	140	140	140	54	0.5	50557.5	306.7
55	102	98	-	55	0.5	90420.2	-
56	110	106	-	56	0.7	60493.2	-
57	110	103	-	57	0.8	51802.5	-
58	74	67	-	58	0.8	10737.5	-

(*Continued*)

Table 21. (*Continued*)

Quality of solution $\tau(\mathcal{B})$				Time (sec.)			
Instance	$\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$	$\mathcal{GE}$	$\mathcal{EG}$	**Instance**	$\mathcal{GG}_{\mathcal{C}}^{\mathcal{R}}$	$\mathcal{GE}$	$\mathcal{EG}$
59	74	74	74	59	0.5	3045.5	2.8
60	100	67	67	60	0.8	1073936.3	623.3
61	42	42	-	61	1.2	300241.7	-
62	832	-	353	62	352.5	-	12552.3

https://doi.org/10.1371/journal.pone.0189283.t021

construction time and the matrix quality (i.e., the number of missing t-combinations). Table 23 summarizes the results of this comparison on $\mathcal{L}_3$. Column 1 lists each set of instances in the benchmark. Columns 2 to 4 present the accumulated solution quality (i.e., the accumulated number of missing t-wise combinations) per set for each strategy. Columns 5 to 7 report the accumulated time per set and strategy.

The experiment reported in this section was conducted to test IPOG-F as an approach for constructing CAs and/or quasi-CAs. The results shown in Table 23 reveal that the matrices constructed using $\mathcal{EG}_{RC}$ have up to 90% fewer missing t-wise combinations than those constructed using IPOG-F. In addition, $\mathcal{EG}_{RC}$ could generate CAs in 40 of the 820 OSCAR instances by reducing the number of missing t-wise combinations to 0, whereas IPOG-F failed to obtained any CA with the desired numbers of rows and columns. Finally, $\mathcal{EG}_{RC}$ achieved better running times than IPOG-F for small values of v and t; however, the time performance of $\mathcal{EG}_{RC}$ rapidly worsened with increasing values of the alphabet size and strength. By contrast, the $\mathcal{GG}_{RC}$ strategy achieved time consumption results similar to those of IPOG-F while also improving the solution quality, making it a better choice than IPOG-F for the construction of quasi-CAs.

Applications of the proposed approaches for solving the OSCAR problem

In this subsection, we demonstrate that the matrices constructed by solving the OSCAR problem can be used as initial matrices for meta-heuristics for CA construction to assist in the construction of better matrices. For this purpose, the outputs of the initialization functions described in the related work section and the best solutions obtained using the proposed OSCAR approaches were used as the initial matrices for a meta-heuristic reported in [22].

Table 24 shows the new upper bounds for $CAN(t, k, v)$ obtained using our proposed methodology. The second column shows the new CA bounds obtained when using the best matrices generated by the proposed OSCAR algorithms as the initial matrices for the meta-heuristic algorithm, and the third column shows the previous upper bounds for those CAs. The best produced solution for each specific instance of the OSCAR problem was used as the initial matrix for the meta-heuristic CA construction algorithm reported in [22], which is also based on the SA algorithm. Because of the small number of missing t-wise combinations in all of the produced initial matrices, the performance of the meta-heuristic algorithm was improved. The results define new upper bounds on $CAN(t, k, v)$ for several CAs.

Conclusions

The present work has indirect implications for the interaction testing of software by aiding in the construction of tests of economical size (a feasible number of test cases). In particular, this paper presents and analyzes strategies for the construction of arrays with sufficiently few

Table 22. Quality of solutions measured as missing t-wise combinations τ, and obtained by the best solution $\mathcal{B}$ from the proposed approaches, and the initialization functions.

Instance	$\tau(\mathcal{B})$	$\tau(\mathcal{S}_1)$	$\tau(\mathcal{S}_2)$	$\tau(\mathcal{S}_3)$	$\tau(\mathcal{S}_4)$
1	267	2892	2188	2834	8820
2	216	2971	2168	2979	9680
3	173	3035	2263	3148	10580
4	154	15106	10357	14731	84270
5	110	77860	62692	78120	347760
6	33	3787	3503	3783	10976
7	60	3504	3251	3553	10192
8	60	5110	4798	5140	15120
9	2	8876	8132	8854	27540
10	74	33573	29110	33682	119880
11	43	3511	3950	3455	4752
12	40	2010	2371	2017	2640
13	187	7501	8252	7410	11648
14	214	6648	7488	6617	10192
15	227	5841	6692	5855	8918
16	226	5046	5797	5071	7644
17	215	4380	5103	4403	6552
18	199	3703	4456	3718	5460
19	175	3109	3843	3087	4550
20	168	2561	3151	2542	3640
21	162	1200	1649	1179	1638
22	155	869	1268	877	1092
23	215	9850	11201	9781	13440
24	139	6541	7807	6552	8820
25	267	3971	4773	3937	5250
26	127	3257	4147	3260	4200
27	74	2021	2601	2033	2520
28	61	1503	2049	1523	1890
29	98	7032	8821	7065	9180
30	9	2979	3957	2941	3672
31	105	25933	29051	25998	55692
32	82	21042	25047	20854	34200
33	142	18960	22757	18893	30780
34	186	16987	20643	16837	27360
35	174	13295	16773	13275	21280
36	363	11720	14674	11791	18620
37	336	10240	13145	10212	15960
38	337	8784	11640	8845	13680
39	320	7435	10103	7475	11400
40	292	6251	8571	6261	9500
41	286	5173	7263	5149	7600
42	339	4233	6102	4207	6080
43	434	3392	5079	3415	4560
44	641	2771	4145	2802	3420
45	108	25471	30741	25577	37800
46	161	20643	25142	20625	30204

(*Continued*)

Table 22. (*Continued*)

Instance	$\tau(\mathcal{B})$	$\tau(\mathcal{S}_1)$	$\tau(\mathcal{S}_2)$	$\tau(\mathcal{S}_3)$	$\tau(\mathcal{S}_4)$
47	173	16203	20203	16201	23520
48	161	12253	15671	12304	17640
49	201	10595	13745	10597	15120
50	140	7450	10182	7432	10500
51	263	4971	7102	5003	6720
52	212	3871	5735	3900	5040
53	140	33977	40780	33997	46200
54	140	19485	23693	19448	25872
55	98	10715	13837	10675	13860
56	106	8840	12033	8937	11550
57	103	7292	9982	7308	9240
58	67	4537	6505	4541	5544
59	74	44732	54182	44972	60720
60	67	23348	29560	23340	30912
61	42	8678	12371	8723	11040
62	**0**	**108240**	**112362**	**113457**	**6314350**

https://doi.org/10.1371/journal.pone.0189283.t022

missing combinations to be considered quasi-CAs. Such arrays are constructed by solving the problem known as the Optimal Shortening of Covering ARrays (OSCAR) problem. The development of these strategies is motivated by the fact that the arrays thus produced can be used as excellent initialization matrices for algebraic or meta-heuristic approaches for the construction of CAs, which are mathematical objects that have broad applications in the testing of software components.

This work presents an analysis of twelve different strategies for solving the OSCAR problem. Five of them correspond to greedy and exact approaches previously described in the literature, whereas the remaining seven algorithms are newly proposed here. The new approaches involve the use of simulated annealing and hybridization in their design. We note that this work also provides pseudocodes for the design of all presented algorithms, including, for the first time, the designs for the greedy approaches, which have been only briefly described in previous works. In addition, to test these strategies, three new OSCAR benchmarks with more than 1,000 instances have been designed, representing a considerable improvement over the previously reported 20-instance benchmark in terms of both size and variety in the values of the strength and alphabet size parameters, t and v, respectively.

The experimental design developed for the comparative analysis involved all three proposed benchmarks. The first benchmark, which consists of small instances, was solved using all twelve strategies: three greedy algorithms $\{\mathcal{GG}_{CR}, \mathcal{GG}_{RC}, \mathcal{GG}_{\substack{R\\C}}\}$, two exact algorithms $\{\mathcal{EE}_{CR}, \mathcal{EE}_{RC}\}$, one meta-heuristic algorithm $\{\mathcal{MM}\}$, and six hybrid approaches $\{\mathcal{GE}, \mathcal{EG}, \mathcal{GM}, \mathcal{MG}, \mathcal{ME}, \mathcal{EM}\}$. Using this benchmark, the algorithms were compared in terms of running time and solution quality (measured as the number of missing t-wise combinations in each constructed array). As expected, the results showed that the greedy algorithms were the fastest, the exact algorithms yielded the best solutions, and the meta-heuristic provided a balance between quality and time. It was also observed that the solution quality of the pure greedy algorithms worsened with increasing instance size, but this situation could be addressed through the use of hybrid algorithms. The hybrid algorithms involving a mixture of greedy and exact approaches had higher running times but also higher solution quality. The

Table 23. Summary of the results of evaluating the performance of algorithms E_1 = IPOG-F, $E_2 = \mathcal{GG}_{\mathcal{RC}}$, and $E_3 = \mathcal{EG}_3$, over the benchmark $\mathcal{L}_3$. The performance is measured in the missing t-wise combinations, and in the time (in seconds) spent to find it.

Instance set	Missings			Time		
	E_1	E_2	E_3	E_1	E_2	E_3
1	1993	237	236	17,18	0,27	0,28
2	16000	27	23	60,46	1,30	8,29
3	87315	6	5	177,25	19,41	446,69
4	228909	1	0	8900,10	450,54	25384,61
5	9501	62	62	38,97	0,28	0,54
6	69913	8	5	219,88	3,22	64,30
7	247416	0	0	2758,92	149,86	21925,19
8	21344	20	20	45,45	0,43	1,24
9	175871	1	0	76,60	7,00	350,69
10	35872	30	30	23,94	0,41	1,90
11	291423	0	0	104,53	18,63	1407,30
12	51142	18	18	32,10	0,62	3,83
13	506926	0	0	176,10	41,27	4483,16
14	325	286	286	17,18	0,13	0,23
15	118	87	87	60,46	1,15	9,88
16	66	58	58	177,25	20,08	469,28
17	61	45	45	8900,10	456,45	32090,95
18	112	108	108	38,97	0,19	0,54
19	69	59	59	219,88	3,32	70,20
20	52	40	40	2758,92	149,98	23156,30
21	84	82	82	45,45	0,29	1,23
22	53	46	46	76,60	7,53	363,88
23	96	90	90	23,94	0,37	3,30
24	47	43	43	104,53	19,80	1488,70
25	63	63	63	32,10	0,54	4,60
26	48	43	43	176,10	42,38	4531,78

https://doi.org/10.1371/journal.pone.0189283.t023

first experiment indicated that hybrid algorithms involving a mixture of the meta-heuristic and greedy strategies are a viable alternative. Such strategies had somewhat higher running times for array construction but resulted in fewer missing t-wise combinations than the hybrid greedy approaches, mainly when the numbers of rows and columns to be deleted were high. In terms of solution quality, the experimental results indicated that the best algorithms were $\mathcal{GE}$ and $\mathcal{EG}$ because they yielded solutions with as few missing t-wise combinations as the exact approaches $\mathcal{EE}_{RC}$ and $\mathcal{EE}_{CR}$ but in less time. In terms of running time, the experiment indicated that the best algorithms were $\mathcal{GG}_{\substack{R\\C}}$ and $\mathcal{GG}_{RC}$ because they were faster than any other algorithms while maintaining an acceptable solution quality; however, we note that for larger instances, the time performance of $\mathcal{GG}_{\substack{R\\C}}$ will be worse than that of $\mathcal{GG}_{RC}$ because it is more strongly affected by the instance size and the numbers of rows and columns to be removed.

The second benchmark was used to perform an in-depth analysis of some of the best strategies, namely, $\mathcal{GG}_{\substack{R\\C}}$, $\mathcal{GE}$ and $\mathcal{EG}$. This experiment tested the performance of these algorithms on larger OSCAR instances to yield a better understanding of their behavior. The best solutions were still produced by the hybrid greedy-exact approaches $\mathcal{GE}$ and $\mathcal{EG}$, but in the latter approach the time increased exponentially. By contrast, the pure greedy algorithm $\mathcal{GG}_{\substack{R\\C}}$

Table 24. New upper bounds.

CAN	New	Previous	CAN	New	Previous
CAN(2, 43, 5)	49	50	*CAN*(355; 2, 12, 18)	355	358
CAN(2, 45, 5)	50	52	*CAN*(354; 2, 8, 18)	354	355
CAN(2, 47, 5)	51	53	*CAN*(2, 28, 18)	497	498
CAN(2, 29, 8)	119	120	*CAN*(2, 20, 20)	510	511
CAN(2, 28, 8)	118	120	*CAN*(2, 19, 20)	509	511
CAN(2, 25, 8)	114	120	*CAN*(2, 18, 20)	508	511
CAN(2, 24, 8)	113	120	*CAN*(2, 16, 20)	507	511
CAN(2, 21, 8)	112	120	*CAN*(2, 15, 20)	502	511
CAN(2, 25, 9)	144	153	*CAN*(2, 14, 20)	501	511
CAN(2, 26, 9)	145	153	*CAN*(2, 13, 20)	499	511
CAN(2, 30, 9)	151	153	*CAN*(2, 12, 20)	497	511
CAN(2, 140, 9)	185	188	*CAN*(2, 11, 20)	495	511
CAN(2, 107, 6)	92	92	*CAN*(2, 10, 20)	491	511
CAN(2, 36, 10)	192	194	*CAN*(2, 9, 20)	482	511
CAN(2, 74, 10)	205	206	*CAN*(2, 8, 20)	465	511
CAN(2, 13, 12)	164	165	*CAN*(2, 7, 20)	429	511
CAN(2, 10, 12)	163	165	*CAN*(2, 20, 21)	519	520
CAN(2, 17, 14)	244	247	*CAN*(2, 18, 21)	518	520
CAN(2, 16, 14)	243	247	*CAN*(2, 16, 21)	517	520
CAN(2, 15, 14)	242	247	*CAN*(2, 14, 21)	516	520
CAN(2, 14, 14)	241	247	*CAN*(2, 13, 21)	514	520
CAN(2, 13, 14)	240	247	*CAN*(2, 11, 21)	513	520
CAN(2, 12, 14)	239	247	*CAN*(2, 9, 21)	501	520
CAN(2, 11, 14)	238	247	*CAN*(2, 8, 21)	499	520
CAN(2, 10, 14)	246	247	*CAN*(2, 21, 22)	525	526
CAN(2, 9, 14)	233	247	*CAN*(2, 16, 22)	524	526
CAN(2, 8, 14)	229	247	*CAN*(2, 12, 22)	523	526
CAN(2, 17, 15)	252	255	*CAN*(2, 11, 22)	522	526
CAN(2, 14, 15)	251	255	*CAN*(2, 10, 22)	521	526
CAN(2, 11, 15)	250	255	*CAN*(2, 8, 22)	520	526
CAN(2, 10, 15)	249	255	*CAN*(2, 22, 24)	621	622
CAN(2, 8, 15)	248	255	*CAN*(2, 16, 24)	620	622
CAN(2, 7, 15)	246	255	*CAN*(2, 10, 24)	619	622

https://doi.org/10.1371/journal.pone.0189283.t024

continued to be fast, and its solutions only slightly deviated from those of the hybrid algorithms. After this analysis, the same benchmark was used to compared the best results from these approaches against the initialization functions generated using state-of-the-art methods. The experimental results showed that in all instances, the number of missing t-wise combinations was reduced by approximately 90% in the matrices constructed using the proposed approach in comparison with those taken from the literature.

Finally, an experiment was conducted using the third benchmark to test IPOG-F as an approach for constructing CAs and/or quasi-CAs. The results revealed that with $\mathcal{EG}_{RC}$, the number of missing t-wise combinations was reduced by up to 90% compared with IPOG-F. Moreover, it was found that $\mathcal{EG}_{RC}$ could obtain CAs in 40 of the 820 OSCAR instances by reducing the number of missing t-wise combinations to 0, whereas IPOG-F failed to obtain any CA with the desired numbers of rows and columns. Finally, it was observed that the running time of $\mathcal{EG}_{RC}$ was better than that of IPOG-F for small values of v and t but worsened

rapidly with increasing values of the alphabet size and strength. By contrast, the $\mathcal{GG}_{RC}$ strategy achieved running times similar to those of IPOG-F while also improving the solution quality, making it a better choice than IPOG-F for the construction of quasi-CAs.

A major drawback of some of the proposed approaches (with the exception of the greedy ones) is the time consumed to solve the problem, which increases with the numbers of rows and columns to be eliminated. Moreover, the experimental design could be improved to test a wider range of possible values to adjust the meta-heuristic and investigate a wider number of strategies. The ranges of values of the alphabet size and strength parameters should be extended to further probe the resulting changes in performance of the different strategies. Future work should also address the lack of an in-depth analysis of the use of the meta-heuristic approach to properly characterize its region of importance. In general, a more extensive characterization study could provide better insight into the behavior of these strategies, and this remains as future work.

Supporting information

S1 Dataset. Benchmark $\mathcal{L}_1$.
(ZIP)

S2 Dataset. Benchmark $\mathcal{L}_2$.
(ZIP)

S3 Dataset. Benchmark $\mathcal{L}_3$.
(ZIP)

Acknowledgments

The authors acknowledge the General Coordination of Information and Communications Technologies (CGSTIC) at CINVESTAV for providing HPC resources on the Hybrid Cluster Supercomputer "Xiuhcoatl", which contributed to the research results reported here. The research reported in this paper was funded through the following projects: CONACYT—Métodos Exactos para Construir Covering Arrays Óptimos, project number 238469; and Cátedras CONACYT—Fortalecimiento de las capacidades de TICs en Nayarit, project number 2143.

Compliance with ethical standards

All authors declare that a) we do not have any conflicts of interest, b) this manuscript is the authors' original work and has not been published nor simultaneously submitted elsewhere, and c) we have acknowledged all entities that have funded this work in any way.

Author Contributions

Conceptualization: Jose Torres-Jimenez.

Formal analysis: Jose Torres-Jimenez, Nelson Rangel-Valdez.

Investigation: Jose Torres-Jimenez.

Methodology: Jose Torres-Jimenez.

Software: Oscar Carrizalez-Turrubiates.

Validation: Jose Torres-Jimenez.

Writing – original draft: Jose Torres-Jimenez, Himer Avila-George, Oscar Carrizalez-Turrubiates.

Writing – review & editing: Jose Torres-Jimenez, Nelson Rangel-Valdez, Himer Avila-George.

References

1. Kuhn DR, Wallace DL, Gallo AM. Software fault interaction and implications for software testing. IEEE Transactions on Software Engineering. 2004; 30(6):418–421. https://doi.org/10.1109/TSE.2004.24
2. Lawrence JF, Kacker RN, Lei Y, Kuhn DR, Forbes M. A survey of binary covering arrays. Journal of Combinatorial Designs. 2011; 18(1):1–30.
3. Jones JA, Harrold MJ. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Transactions on software Engineering. 2003; 29(3):195–209. https://doi.org/10.1109/TSE.2003.1183927
4. Sloane NJA. Covering arrays and intersecting codes. Journal of Combinatorial Designs. 1993; 1(1):51–63. https://doi.org/10.1002/jcd.3180010106
5. Bush KA. Orthogonal arrays of index unity. Annals of Mathematical Statistics. 1952; 23(3):426–434. https://doi.org/10.1214/aoms/1177729387
6. Colbourn CJ, Dinitz JH. The CRC handbook of combinatorial designs. CRC Press; 1999.
7. Colbourn CJ. Combinatorial aspects of covering arrays. Le Matematiche. 2004; 58:121–167.
8. Meagher K. Non-isomorphic generation of covering arrays. University of Regina; 2002.
9. Lopez-Escogido D, Torres-Jimenez J, Rodriguez-Tello E, Rangel-Valdez N. Strength Two Covering Arrays Construction Using a SAT Representation. In: Gelbukh A, Morales EF, editors. MICAI 2008: Advances in Artificial Intelligence. vol. 5317 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2008. p. 44–53.
10. Bracho-Rios J, Torres-Jimenez J, Rodriguez-Tello E. A New Backtracking Algorithm for Constructing Binary Covering Arrays of Variable Strength. In: MICAI 2009: Advances in Artificial Intelligence. vol. 5845 of Lecture Notes in Computer Science. Springer; 2009. p. 397–407.
11. Banbara M, Matsunaka H, Tamura N, Inoue K. Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers. In: Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning. Springer-Verlag; 2010. p. 112–126.
12. Hartman A, Raskin L. Problems and algorithms for covering arrays. Discrete Mathematics. 2004; 284 (1–3):149–156. https://doi.org/10.1016/j.disc.2003.11.029
13. Martirosyan S, Trung TV. On t-Covering Arrays. Designs, Codes and Cryptography. 2004; 32(1–3):323–339. https://doi.org/10.1023/B:DESI.0000029232.40302.6d
14. Chateauneuf M, Kreher DL. On the state of strength-three covering arrays. Journal of Combinatorial Design. 2002; 10(4):217–238. https://doi.org/10.1002/jcd.10002
15. Colbourn CJ, Martirosyan SS, Mullen GL, Shasha D, Sherwood GB, Yucas JL. Products of mixed covering arrays of strength two. Journal of Combinatorial Design. 2006; 14(2):124–138. https://doi.org/10.1002/jcd.20065
16. Cohen DM, Dalal SR, Fredman ML, Patton GC. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering. 1997; 23(7):437–444. https://doi.org/10.1109/32.605761
17. Tung YW, Aldiwan WS. Automating test case generation for the new generation mission software system. In: IEEE Aerospace Conference Proceedings. vol. 1. IEEE Computer Society; 2000. p. 431–437.
18. Bryce RC, Colbourn CJ, Cohen MB. A framework of greedy methods for constructing interaction test suites. In: Proceedings of the 27th International Conference on Software Engineering. ICSE'05; 2005. p. 146–155.
19. Forbes M, Lawrence J, Lei Y, Kacker RN, Kuhn DR. Refining the In-Parameter-Order Strategy for Constructing Covering Arrays. Journal of Research of the National Institute of Standards and Technology. 2008; 113(5):287–297. https://doi.org/10.6028/jres.113.022 PMID: 27096128
20. Colbourn CJ, Cohen MB, Turban R. A Deterministic Density Algorithm for Pairwise Interaction Coverage. In: Proceedings of the IASTED International Conference on Software Engineering; 2004. p. 242–252.
21. Shiba T, Tsuchiya T, Kikuno T. Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. In: Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004. vol. 1. IEEE Computer Society; 2004. p. 72–77.

22. Avila-George H, Torres-Jimenez J, Gonzalez-Hernandez L, Hernández V. Metaheuristic approach for constructing functional test-suites. IET Software. 2013; 7(2):104–117. https://doi.org/10.1049/iet-sen.2012.0074

23. Nurmela KJ. Upper bounds for covering arrays by tabu search. Discrete Applied Mathematics. 2004; 138(1–2):143–152. https://doi.org/10.1016/S0166-218X(03)00291-9

24. Carrizales-Turrubiates O, Rangel-Valdez N, Torres-Jimenez J. Optimal Shortening of Covering Arrays. In: Batyrshin I, Sidorov G, editors. Advances in Artificial Intelligence. vol. 7094 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2011. p. 198–209.

25. Feige U. A threshold of *ln n* for approximating set cover. Journal of the ACM. 1998; 45(4):634–652. https://doi.org/10.1145/285055.285059

26. Cohen DM, Colbourn CJ, Ling ACH. Constructing strength three covering arrays with augmented annealing. Discrete Mathematics. 2008; 308(13):2709–2722. https://doi.org/10.1016/j.disc.2006.06.036

27. Rodriguez-Tello E, Torres-Jimenez J. Memetic Algorithms for Constructing Binary Covering Arrays of Strength Three. In: Collet P, Monmarch N, Legrand P, Schoenauer M, Lutton E, editors. Artifical Evolution. vol. 5975 of Lecture Notes in Computer Science. Springer; 2010. p. 86–97.

28. Colbourn CJ. Tables of Covering Arrays; 2017. OnLine. http://www.public.asu.edu/~ccolbou/src/tabby/catable.html

29. National Institute of Standards and Technology. Tables of Covering Arrays; 2017. OnLine. http://math.nist.gov/coveringarrays/ipof/ipof-results.html

30. Rényi A. Foundations of Probability. Wiley; 1971.

31. Kleitman DJ, Spencer J. Families of k-independent sets. Discrete Mathematics. 1973; 6(3):255–262. https://doi.org/10.1016/0012-365X(73)90098-8

32. Cohen DM, Dalal SR, Parelius J, Patton GC. The combinatorial design approach to automatic test generation. IEEE Transactions on Software Engineering. 1996; 13(5):83–88. https://doi.org/10.1109/52.536462

33. Lei Y, Kacker RN, Kuhn DR, Okun V, Lawrence J. IPOG: A General Strategy for T-Way Software Testing. In: ECBS'07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. IEEE Computer Society; 2007. p. 549–556.

34. Bryce RC, Colbourn CJ. A density-based greedy algorithm for higher strength covering arrays. Software Testing, Verification and Reliability. 2009; 19(1):37–53. https://doi.org/10.1002/stvr.393

35. Quiz-Ramos P, Torres-Jimenez J, Rangel-Valdez N. Constant Row Maximizing Problem for Covering Arrays. In: Artificial Intelligence, 2009. MICAI 2009. Eighth Mexican International Conference on. IEEE Computer Society; 2009. p. 159–164.

36. Nayeri P, Colbourn CJ, Konjevod G. Randomized post-optimization of covering arrays. European Journal of Combinatorics. 2013; 34(1):91–103. https://doi.org/10.1016/j.ejc.2012.07.017

37. Lara-Alvarez C, Avila-George H. New Algorithm for Post-Processing Covering Arrays. International Journal of Advanced Computer Science and Applications. 2015; 6(12):250–254. https://doi.org/10.14569/IJACSA.2015.061234

38. Walker RA II, Colbourn CJ. Tabu search for covering arrays using permutation vectors. Journal of Statistical Planning and Inference. 2009; 139(1):69–80. https://doi.org/10.1016/j.jspi.2008.05.020

39. Avila-George H, Torres-Jimenez J, Hernández V. New bounds for ternary covering arrays using a parallel simulated annealing. Mathematical Problems in Engineering. 2012; 2012:1–18. https://doi.org/10.1155/2012/897027

40. Gonzalez-Hernandez L, Rangel-Valdez N, Torres-Jimenez J. Construction of mixed covering arrays of strengths 2 through 6 using a tabu search approach. Discrete Mathematics, Algorithms and Applications. 2012; 04(03):1–20. https://doi.org/10.1142/S1793830912500334

41. Bryce RC, Colbourn CJ. One-test-at-a-time Heuristic Search for Interaction Test Suites. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. GECCO'07; 2007. p. 1082–1089.

42. Bao X, Liu S, Zhang N, Dong M. Combinatorial Test Generation Using Improved Harmony Search Algorithm. International Journal of Hybrid Information Technology. 2015; 8(9):121–130. https://doi.org/10.14257/ijhit.2015.8.9.13

43. Donald LK, Stinson DR. Combinatorial algorithms: generation, enumeration, and search. CRC Press; 1999.

44. Torres-Jimenez J, Rangel-Valdez N, Kacker RN, Lawrence JF. Combinatorial Analysis of Diagonal, Box, and Greater-Than Polynomials as Packing Functions. Applied Mathematics & Information Sciences. 2015; 9(6):2757–2766.

45. Van Laarhoven PJM, Arts EHL. Simulated Annealing: Theory and Applications. Philips Research Laboratories; 1992.

46. Rangel-Valdez N, Torres-Jimenez J, Bracho-Rios J, Quiz-Ramos P. Problem and Algorithm Fine-Tuning—A Case of Study using Bridge Club and Simulated Annealing. In: Correia AD, Rosa AC, Madani K, editors. IJCCI; 2009. p. 302–305.

47. Pérez Espinosa H, Avila-George H, Rodríguez-Jacobo J, Cruz-Mendoza HA, Martínez-Miranda J, Edrein Espinosa-Curiel I. Tuning the Parameters of a Convolutional Artificial Neural Network by Using Covering Arrays. Research in Computing Science. 2016; 121:69–81.